Results on the computational power of polarizationless P systems with active membranes

PhD Thesis

Károly Hajagos
Supervisor: Dr. Zsolt Gazdag, PhD

Doctoral School of Computer Science
Department of Foundations of Computer Science
Faculty of Science and Informatics
University of Szeged

Szeged
2025

Contents

Chapter 1

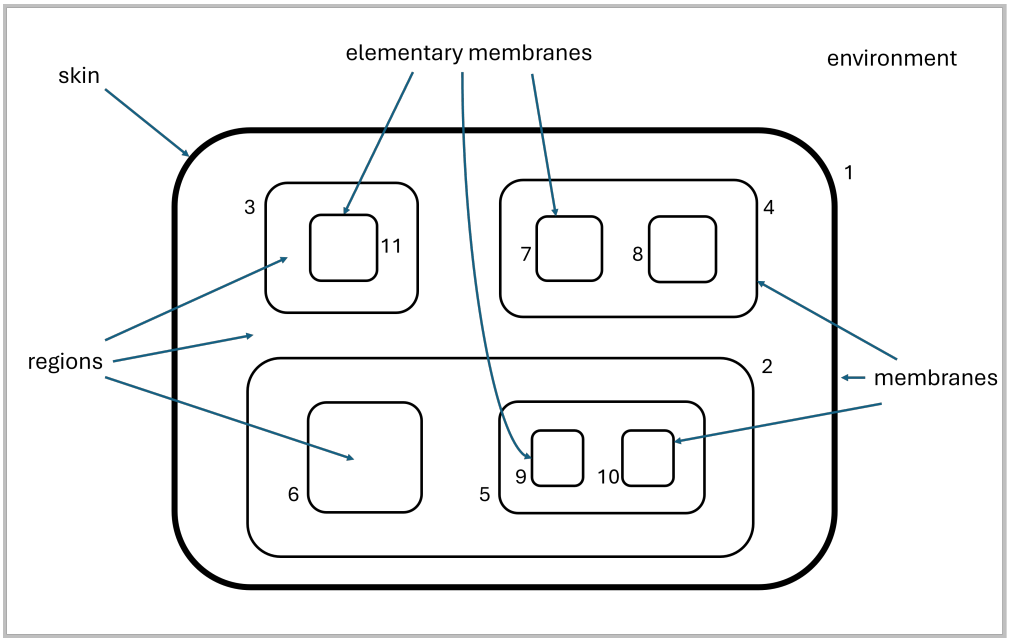
Introduction

Membrane computing [39, 43] is a branch of natural computing inspired by the architecture and functioning of living cells. The aim of membrane computing is to discover and formulate computational models to simulate the behavior of biological cells. These models are called *membrane systems*, which are also known as *P systems* after Gheorghe Păun, who initiated membrane computing in 1998. P systems are universal computational models in the sense that they are able to compute all recursively enumerable sets of natural numbers [39].

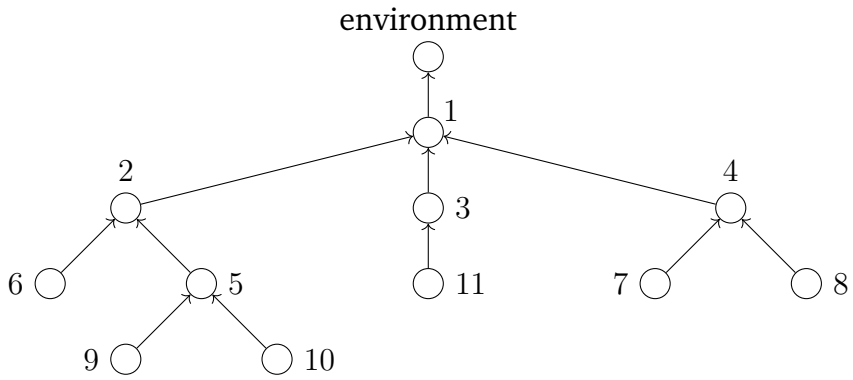
Initially, as an underlying structure on which the membrane systems were based, a hierarchical (cell-like) arrangement of a finite number of *membranes* called *membrane structure* was considered. Membrane structures can be portrayed by properly embedded squared parentheses, a tree or a Venn-diagram as Figure 1.1 shows. Throughout the thesis, we consider membrane structures mainly as trees. The outermost membrane of the structure is called the *skin membrane* that is surrounded by the *environment*. The innermost membranes of the structure, in which no other membrane is embedded, are called *elementary membranes*. For any membrane x, excluding the skin membrane, the *outer membrane* (or *parent membrane*) of x is the membrane x is immediately embedded in. Analogously, for any membrane x excluding the elementary membranes, the *inner membranes* (or *child membranes*) of x are the membranes that are immediately embedded in x. Each membrane has a *membrane label* for identification purposes and we can assume that initially these labels are assigned to the membranes uniquely from a given set of labels. In Figure 1.1, the membranes are labeled with numbers.

In biology, membranes demarcate compartments that may contain some chemicals (i.e. molecules). In the abstract model, the compartments are called *membrane regions* and the chemicals are called *objects*. More precisely, each region and the environment may contain some *copies* (or *instances*) of each object. That is, the content of each region and the content of the environment are associated with a *multiset* (i.e. a set with multiplicities) of objects. This gives us the definition of a *membrane configu-*

ration of the P system which is the membrane structure with a multiset assignment to every region. For convenience reasons, we represent each multiset with some words. As an example, if a, b and c are objects, then a multiset that consists of two copies of a, three copies of b and a copy of c can be represented by the word a^2b^3c.



(a)



(b)

$$\Big[\Big[\Big[\,\big]_9\big[\,\big]_{10}\Big]_5\Big[\,\big]_6\Big]_2\Big[\Big[\,\big]_{11}\Big]_3\Big[\Big[\,\big]_7\Big[\,\big]_8\Big]_4\Big]_1$$

(c)

Figure 1.1: *A membrane structure portrayed by a Venn-diagram (a), a tree (b) and properly embedded squared parentheses (c)*

In the membrane regions (or simply regions) the objects may evolve in accordance with the *rules* of the P system, simulating the biological processes. Basically, these rules can be *multiset-rewriting rules* that represent the reactions between the chemicals, *communication rules* that allow the objects to pass through the membranes, providing the possibility of communication between the regions, or rules that are able to modify the membrane structure by removing membranes or adding new membranes.

P systems are *parallel* computational models, which means that in different membranes different operations can be executed (almost) independently. These operations are rule applications that go on in a synchronous way in the time units provided by a global clock which holds for all membrane regions. In each time unit, the appropriate rules, chosen in a *nondeterministic* manner, are employed resulting in the transformation of one configuration to another. A *computation* of the P system is the sequence of these transformations, and the transformations between configurations are called *transitions* or *computation steps*. When defining P systems, an *initial configuration* is considered, where the computation starts from. We say that a computation is *halting* if it reaches a configuration in which no rules can be applied anymore. The *result* (or the *output*) of a halting computation is the multiset associated with the region of a designated *output membrane* or the environment, in the last configuration.

P systems are usually used in *generating* mode or *recognizing* mode. When the P system is in generating mode, that is, generates a set of objects, its functioning resembles the use of a grammar in formal language theory. In this case, the result of a computation can be the number of object occurrences in the region of the output membrane. If it is necessary to distinguish the objects, the result can be a vector containing the multiplicities of each object. Another approach takes a word as a result of the computation that can be read in several ways. As an example, the order of appearance of object copies in the region of the output membrane can determine the order of the objects in this word. When several objects appear simultaneously in this membrane region, all permutations of these objects are permitted.

When a P system is in recognizing mode, that is, recognizes a language, it works like an automaton in theory of computation. In the field of complexity theory, a decider for a language L is typically a Turing machine M that halts on each input and produces an output that is either *accepted* or *rejected*. The output for a word w is accepted if and only if $w \in L$. M may determine the output in several equivalent ways, such as halting in a designated *accept* or *reject* state, halting in either a *final* or *non-final* state, or by writing the output on the output tape. In membrane computing, the approach to solving decision problems mirrors that of complexity theory: the P system eventually halts, accepting or rejecting the *input* which is encoded by the *input multiset* of objects. The P system has a designated *input membrane* that contains this multiset initially. Typically, P systems indicate their decision by producing a

specific symbol, yes or no, in the environment or in a designated output membrane. The initial formalization of these conditions appeared in [47], where the concept of *accepting P systems* was introduced. A P system Π is an accepting P system if it has only halting computations and its object alphabet includes symbols yes and no. Furthermore, for any computation of Π, it is required that the object yes or the object no is sent out to the environment, but not both. This definition was further refined in the concept of *recognizer P systems*, which stipulates that exactly one yes or no must be sent out to the environment or to a designated output membrane, exclusively in the last step of the computation (see, e.g., [45]). Recognizer P systems are often considered *confluent*, which means that all halting computations on the same input must produce the same output. More precisely, given an instance I of a decision problem L, a recognizer P system Π deciding L should halt on I with the correct output yes or no, depending on whether I is a positive instance of L or not. That is, even if Π can have many different computations on I, all of these computations should halt with the same output. Throughout the thesis, we consider only confluent P systems. If a confluent P system is such that each of its reachable configurations has at most one successor, the P system is said to be *deterministic*.

Now, we take a brief look at three of the most well known types of cell-like P systems: transition P systems, P systems with symport/antiport rules and P systems with active membranes. We only mention here that several other P system types have been introduced and investigated in membrane computing, e.g., P Automata [12] or Spiking Neural P systems [20]. As the thesis is specifically based on polarizationless P systems with active membranes, that variant will be discussed in detail in Chapter 2.

1.1 Transition P systems

A transition P system [39, 46] works with *evolution rules* of the form $u \to v$, where $u \in O^*$ with O denoting the alphabet of objects and $v = v'$ or $v = v'\delta$ with $v' \in (O \times \{here, out, in\})^*$. Thus, the elements of v' are of the form (a, dir) where $a \in O$ and $dir \in \{here, out, in\}$. If $|u| > 1$, then the objects can *cooperate* with each other, which makes the P system *cooperative*. The region of every membrane has a set of rules R_h, where h is the label of the membrane. For each rule associated with the skin membrane, $v = v'$ must hold.

The semantics of these rules are the following. Let x and y be two membranes with y being the parent membrane of x, and let h be the label of x. Moreover, let $\mathbf{m}_x$ and $\mathbf{m}_y$ be the multisets associated with the regions of x and y, respectively. A rule $r = u \to v$ is applicable in the region of x only if R_h contains r, and $\mathbf{m}_x$ includes the multiset represented by u. If $v = v'$ then the application of r consists of two substeps, if $v = v'\delta$ then the application has an additional substep. In the

first substep, each copy of the objects in u is removed from $\mathfrak{m}_x$. Then, in the second substep, the following happens for each $a \in O$ occurring in v. For each appearance of

- $(a, here)$: $\mathfrak{m}_x$ gets extended by a copy of a,
- (a, out): $\mathfrak{m}_y$ gets extended by a copy of a, and
- (a, in): the multiset of a nondeterministically chosen inner membrane of x gets extended by a copy of a.

If $v = v'\delta$ then after the execution of these two substeps, x gets *dissolved* which means that the membrane bordering its region is removed resulting in the fusion of the regions of x and y. That is, $\mathfrak{m}_y$ gets extended by $\mathfrak{m}_x$, moreover, after the dissolution, the membrane structure changes the following way. The membrane x disappears permanently and y becomes the parent membrane of the child membranes of x. The skin membrane never gets dissolved. Since there is no set of rules associated with the environment, the objects that are sent out from the skin membrane never come back from the environment.

In every computation step, the rules are applied according to an application mode. The most proposed mode is *maximal parallelism*, which means the following. Each applicable rule can be chosen for application multiple times in a nondeterministic manner, and the chosen rules are applied collectively in parallel to the maximum extent possible. This functioning is motivated by biology, where the biochemical reactions take place in parallel.

We demonstrate maximal parallelism on a small example. Consider the membrane configuration shown on the left-hand side of Figure 1.2 and let $R_2 = \{r_1 =$

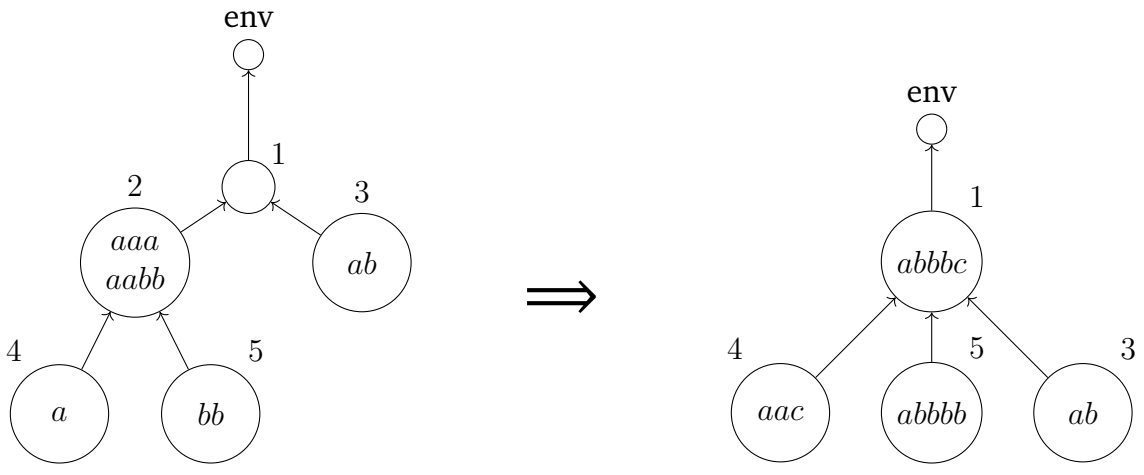


Figure 1.2: *A possible application of the rules that is associated with the membrane with label* 2*. The nodes denote membranes and the root denotes the environment. The multiset associated with the regions are written inside the nodes.*

$aa \to (c, in)(a, here)\delta, r_2 = a \to (b, out)(b, in)(a, in), r_3 = ab \to (c, here)\}$. For the sake of simplicity, we assume that the sets of rules associated with the other membrane regions are empty. Then, during a computation step, applying r_1 once, r_2 twice, and r_3 once is a possible behavior. The configuration we obtain in this way is shown on the right-hand side of Figure 1.2.

We note that, for each membrane with a label h, a partial order relation ρ_h over R_h is often considered in transition P systems, which defines priority on the rules. In this case, during the application of the rules, ρ_h must be respected. We only mention that a *catalytic transition P system* is a special type of transition P systems whose rules are of the form $ca \to cv$ where $c \in C, a \in O - C$ and $v \in (O - C)^*$ with $C \subset O$ being the set of *catalysts*.

Catalytic transition P systems with priority are usually used in generating mode and they are able to generate all recursively enumerable sets of natural numbers making them computationally complete models [39].

1.2 P systems with symport/antiport rules

P systems with symport/antiport rules [41, 42] are communicating P systems, that is, the rules of the membrane system provide only the communication between membrane regions. This communication simulates the transportation of chemicals through the membranes. There are three types of rules in these P systems ($a, b \in O$, again, O denotes the alphabet of objects):

- *symport rules* of the forms (ab, in) or (ab, out), which represent that two molecules pass together through the same membrane to the same direction;
- *antiport rules* of the form $(a, in; b, out)$, which represent that two molecules pass through the same membrane at once but in opposite directions;
- *uniport rules* of the forms (a, in) or (a, out), which represent that a molecule passes through a membrane alone.

There is a set of rules R_h associated with the region of each membrane with label h. As in transition P systems, the usual application mode in P systems with symport/antiport rules is the maximally parallel mode which works similarly. We note that objects may pass through even the skin membrane, meaning that objects appearing in the environment may come back to the membrane regions. We also note that symport and antiport rules can be generalized by considering the symport rules of the forms (u, in) or (u, out) and the antiport rules of the form $(u, in; v, out)$ for some $u, v \in O^*$.

P systems with symport/antiport rules are computationally complete models as they are able to simulate Turing machines [42].

1.3 P systems with active membranes

The P system types that we briefly introduced above cannot increase the number of membranes in their membrane structure during their functioning. However, creating new membranes is quite substantiated by biology; e.g. division of membranes is a common operation in living cells. This idea motivated the development of P systems with active membranes that simulates this kind of behavior. The concepts of P systems with active membranes were first presented by Gheorghe Păun [40].

Basically, along with the other type of rules, P systems with active membranes work with *division rules for elementary membranes*. During a division, the membrane is duplicated such that the copy membrane has the same parent membrane and its content is almost the same; only the objects occurring in the applied rule may differ. Using these rules, the P systems can create an exponential number of elementary membranes in linear time. Besides membrane division, the other innovation in P systems with active membranes is the *electrical polarization* (or *charge*, or simply *polarization*) of the membranes. Each membrane possesses a charge which can be *positive* (+), *negative* (-) or *neutral* (0). The concept of polarizations is also motivated by biochemistry: cellular membranes can be active in the sense that they can interact with chemicals that pass through them by using protein channels. During this interaction, both the chemicals and the membranes can be modified. In the latter case, the polarization of the membrane can change. In the formal model, polarizations can be used to control the computations appropriately, which provides a higher degree of fine-tuning of the P system. Moreover, with this feature, the P system is able to store some information about the operations executed on the membranes.

Now, we present the rules of P systems with active membranes. Let O and H be the alphabet of objects and the set of membrane labels, respectively. P systems with active membranes work with the following type of rules (where $h \in H$, $a, b, c \in O$ and $\alpha, \alpha_1, \alpha_2, \alpha_3 \in \{+, -, 0\}$ are polarizations)

(a) $[a \to v]_h^\alpha$ (*object evolution* rules);

(b) $a[\ ]_h^{\alpha_1} \to [b]_h^{\alpha_2}$ (*in-communication* rules);

(c) $[a]_h^{\alpha_1} \to [\ ]_h^{\alpha_2} b$ (*out-communication* rules);

(d) $[a]_h^\alpha \to b$ (*membrane dissolution* rules);

(e) $[a]_h^{\alpha_1} \to [b]_h^{\alpha_2}[c]_h^{\alpha_3}$ (*division* rules for elementary membranes).

In contrast with transition P systems and P systems with symport/antiport rules, in P systems with active membranes each rule contains exactly one object on the left-hand side. Object evolution rules (type (a)) are rewriting rules that can be applied only in a membrane with a label h and with an appropriate polarization, whose region contains a copy of object a. Notice that object evolution rules (or simply

evolution rules) do not directly involve the membranes. An in-communication rule (type (b)) is employable on a copy of a only if the membrane whose region contains this copy has a child membrane with a label h and an appropriate polarization. Rules of types (c), (d) and (e) are applicable only if there is a membrane with label h and an appropriate charge, whose region contains a copy of a. The skin membrane cannot be dissolved. Note that applying evolution rules never changes the polarization of the membrane, unlike the application of the other rules. However, this could lead to undesirable behavior: the application of at least two non-evolution rules on a membrane could modify the polarization differently. Consequently, each membrane can be the subject of at most one rule of type (b), (c), (d), or (e) at the same time.

P systems with active membranes also work in a maximally parallel manner: at each computation step, the system first nondeterministically assigns appropriate rules to the instances of objects of the current membrane configuration such that the assigned multiset S of rules satisfies the following properties: (i) at most one rule from S is assigned to any instance of an object in the membrane configuration, (ii) a membrane can be the subject of at most one rule of types (b)-(e) in S, and (iii) S is maximal, that is, it cannot be expanded by addition of any rule without breaching either condition (i) or condition (ii). Then, the rules in S are applied in a bottom-up manner during the computation step.

The semantics of the rules will be thoroughly discussed in Section 2.2, where we consider the rules without polarizations. We note that *division rules for non-elementary membranes* of the form

$$\left[\left[\right]_{h_1}^{\alpha_1} \cdots \left[\right]_{h_k}^{\alpha_1} \left[\right]_{h_{k+1}}^{\alpha_2} \cdots \left[\right]_{h_n}^{\alpha_2} \right]_{h_0}^{\alpha_0} \rightarrow \left[\left[\right]_{h_1}^{\alpha_3} \cdots \left[\right]_{h_k}^{\alpha_3} \right]_{h_0}^{\alpha_5} \left[\left[\right]_{h_{k+1}}^{\alpha_4} \cdots \left[\right]_{h_n}^{\alpha_4} \right]_{h_0}^{\alpha_6}$$

where $n \in \mathbb{N}, 1 \leq k \leq n, h_i \in H, 0 \leq i \leq n$ and $\alpha_0, \ldots, \alpha_6 \in \{+, -, 0\}$ with $\{\alpha_1, \alpha_2\} = \{+, -\}$, are often considered, but we do not view them as part of the basic model. These rules provide the possibility of duplicating subtrees of the membrane structure. P systems with active membranes using division rules for non-elementary membranes are computationally universal models [40].

P systems with active membranes are usually used in recognizing mode, that is, in most cases they are used to solve decision problems. Throughout the thesis, we investigate only P systems with active membranes functioning in recognizing mode.

We will now discuss some main results achieved in the field of P systems with active membranes. It turned out quite early that these systems working in polynomial time can solve **NP**-complete problems [23, 40, 58]. For example, in [40], the **NP**-complete SAT problem was solved in linear time by P systems with active membranes. In the solution, the initial membrane structure consists of linearly nested membranes. The elementary membrane division rules are used to create 2^n elementary membranes in n steps where n is the number of variables in the input formula.

That is, each elementary membrane is associated with a possible truth assignment of the variables. At the same time, the P system creates a branch in the tree for each truth assignment by using non-elementary membrane division rules. Then, the evaluation of the clauses is carried out by sending the information from the leaves of the tree towards the environment by using membrane dissolution.

When using division only for elementary membrane, their computational power characterizes the complexity class $\mathbf{P}^{\#\mathbf{P}}$, that is, the class of all problems solved by polynomial-time deterministic Turing machines with polynomial-time counting oracles [25]. Moreover, if not only elementary membranes but also non-elementary membranes can be divided, then the computational power of these P systems presumably rises as in this case they can solve in polynomial time exactly the problems in **PSPACE** [9, 50, 52]. Interestingly, removing the polarizations of the membranes does not affect the computational power of these P systems when non-elementary membrane division is allowed [11]. For recent surveys on the computational power of P systems with active membranes, we refer to [36, 51] and the references therein.

In the thesis, we deal only with *polarizationless P systems with active membranes*. This restriction can be achieved by considering P systems with active membranes whose membranes have the same polarization that never changes. Hence, they are not taken into account neither at the membranes nor in the rules. On the other hand, in polarizationless P systems with active membranes, the membranes can still be the subject of at most one non-evolution rule at once in a computation step.

It is still an open question how exactly the removal of polarizations affects the computational power in P systems with active membranes when division is allowed only for elementary membranes. In the work [11] we mentioned above, the authors considered P systems in which non-elementary membrane division was allowed but the use of polarizations was not, and it was shown that this variant is still powerful enough to solve the **PSPACE**-complete QSAT problem (deciding whether a given Quantified Boolean formula is true or not) in polynomial time. On the other hand, it is still open whether **NP**-complete problems can be solved in polynomial time without polarization using only elementary membrane division. In fact, Păun conjectured already in 2005 that polynomial-time P systems without non-elementary membrane division rules cannot solve all problems in **NP** if we remove the polarizations of the membranes. Since then, this conjecture has been known as *Păun's conjecture* (which is often called the *P conjecture* in the literature) which sounds as follows [44]:

> "*Can the polarizations be completely avoided? (This means to have all membranes, always, of the same polarization, which amounts to say that this parameter will play no role in the system functioning.) The feeling is that this is not possible – and such a result would be rather sound: passing from no polarization (which, in fact, means one polarization) to two polarizations amounts to passing from non-efficiency to efficiency.*"

However, proving this conjecture is challenging, as even a polarizationless P system can produce exponentially many membranes and exponentially many objects in the membrane regions in linear time. Nevertheless, there are several partial solutions to Păun's conjecture, see, for example, [16, 19, 27, 31, 56].

1.4 The structure of the thesis

In this section, we present the essence of the chapters the thesis is composed of.

In Chapter 2, we discuss those fundamental components that will be necessary in the following chapters. First, we define multisets, graphs, the basics of propositional logic, and some important classical computational complexity classes. Then, in the second section of the chapter, we introduce polarizationless P systems with active membranes. Along with the definition of such a P system, we describe in detail how a computation step is executed. In the third and final section of the chapter, we give the formal definitions of recognizer P systems and uniform families of P systems.

According to [19], without dissolution rules, P systems with active membranes cannot solve any computationally hard problem in polynomial time even if the use of non-elementary membrane division is permitted. In fact, if we omit polarizations, the computational power of these P systems weakens significantly [35]. These results suggest that dissolution rules play an important role in polarizationless P systems with active membranes. This motivates the idea of studying what these P systems are capable of regarding their computational power, when only dissolution rules are allowed to be applied. In Chapter 3, we show that polynomial-time polarizationless P systems with active membranes, working only with dissolution rules and using $\mathbf{L}$-uniformity conditions, are able to solve the $\mathbf{NL}$-complete REACHABILITY problem. This gives an $\mathbf{NL}$ lower bound to the computational power of these P systems.

In Chapter 4, we continue the discussion on polarizationless P systems with active membranes that use only dissolution rules. We consider a variant of the HORNSAT problem called HORN3SATNORM which is also a $\mathbf{P}$-complete problem. We will show that these P systems, working in polynomial time, can solve HORN3SATNORM. This means that they could solve everything that polarizationless P systems can if Păun's conjecture is true. As a consequence, we could conclude that allowing the use of all the other classical rules does not raise the computational power of the resulting P system. In addition, this result means that polarizationless P systems with active membranes using only dissolution rules characterize $\mathbf{P}$. In this chapter, we give a method for constructing an appropriate uniform family of P systems to achieve this result, while paying attention to defining very tight uniformity conditions.

It is known that polarizationless P systems with active membranes can solve $\mathbf{PSPACE}$-complete problems in polynomial time without using in-communication rules but using the classical (also called strong) non-elementary membrane divi-

sion rules [11]. In Chapter 5, we show that this holds also when in-communication rules are allowed but strong non-elementary division rules are replaced with weak non-elementary division rules, a type of rules which is an extension of elementary membrane division rules to non-elementary membranes. It is known that without in-communication rules, and even with polarizations, these P systems can solve in polynomial time only the problems in $\mathbf{P}^{\mathbf{NP}}$ [26], which is the class of problems solvable by polynomial-time deterministic Turing machines with $\mathbf{NP}$ oracles. Hence, our result proves that in-communication rules serve as a borderline between $\mathbf{P}^{\mathbf{NP}}$ and $\mathbf{PSPACE}$ regarding the computational power of these P systems.

In Chapter 6, we consider a variant of P systems called elimination P systems. These are polarizationless P systems with active membranes extended with rules of the form $[ab \rightarrow \varepsilon]_h$, where a and b are objects and ε denotes the empty word. The semantics of these rules are as follows: when a and b are present together within the same membrane with label h, they are both eliminated and no new objects are produced. We investigate the computational power of two types of elimination P systems designed to solve decision problems: recognizer elimination P systems and extended acknowledger elimination P systems. In the more general extended acknowledger elimination P systems, an accepting computation should produce one or more copies of yes, whereas a rejecting computation should not produce any copies of yes. The main result in this chapter is that while polynomial-time recognizer elimination P systems with no dissolution rules can solve only problems in $\mathbf{NL}$, extended acknowledger elimination P systems with no dissolution rules are able to solve $\mathbf{PP}$-complete problems in polynomial time. This demonstrates the importance of accepting conditions in these P systems.

Chapter 2

Preliminaries

In this chapter, we recall the fundamentals and the necessary notations of multisets, graphs and propositional logic, along with some essential computational complexity classes. However, we assume that the reader is familiar with the basic concepts of these subjects (for comprehensive guides to the topics, see, e.g. [18, 38, 53]). Then, we give a formal definition of polarizationless P systems with active membranes and demonstrate how a computation step works. In the last section, we formally present recognizer P systems that are used to decide languages, then, we define uniform families of membrane systems that are usually devoted to solve decision problems.

2.1 Basic terminology

Denote $\mathbb{N}$ the set of natural numbers including zero. For numbers $i \leq j$ in $\mathbb{N}$, $[i, j]$ denotes the set $\{i, i+1, \ldots, j\}$ and, for $j \geq 1$, $[j]$ denotes the set $[1, j]$.

2.1.1 Multisets

Let Σ be an alphabet of symbols and let u be a word over Σ. The *length* of u is the number of occurrences of the symbols in u, which is denoted by $|u|$. If $|u| = 0$, then u is the *empty word* that we denote by ε. Moreover, denote $|u|_a$, for any $a \in \Sigma$, the number of occurrences of a in u.

A *multiset* over Σ is a mapping $\mathfrak{m} : \Sigma \to \mathbb{N}$ and for a symbol $a \in \Sigma$, $\mathfrak{m}(a)$ is the number of occurrences of a in $\mathfrak{m}$. If $\mathfrak{m}(a) > 0$, then we say that $\mathfrak{m}$ *contains* a copy of a, which is denoted by $a \in \mathfrak{m}$. Each multiset is represented by some words as follows. The word $u \in \Sigma^*$ *represents* $\mathfrak{m}$ if for each $a \in \Sigma$, $\mathfrak{m}(a)$ equals $|u|_a$. Hence, for the sake of simplicity, for any $u \in \Sigma^*$, we often define $\mathfrak{m}_u$ as $\mathfrak{m}_u = u$. In fact, we often use u instead of $\mathfrak{m}_u$ when it is more convenient. The set of all multisets over Σ is denoted by $\boldsymbol{\mathcal{M}}(\Sigma)$.

For two multisets $\mathfrak{m}_1$ and $\mathfrak{m}_2$ over Σ, we say that $\mathfrak{m}_2$ is *included* in $\mathfrak{m}_1$, denoted by $\mathfrak{m}_2 \subseteq \mathfrak{m}_1$, if for each $a \in \Sigma$, $\mathfrak{m}_2(a) \leq \mathfrak{m}_1(a)$. The *difference* $\mathfrak{m}_1 - \mathfrak{m}_2$ of $\mathfrak{m}_1$ and $\mathfrak{m}_2$ is defined if $\mathfrak{m}_2 \subseteq \mathfrak{m}_1$, in which case $\mathfrak{m}_1 - \mathfrak{m}_2$ is a multiset $\mathfrak{m}_3$ with $\mathfrak{m}_3(a) + \mathfrak{m}_2(a) = \mathfrak{m}_1(a)$ for each $a \in \Sigma$. The *sum* of $\mathfrak{m}_1$ and $\mathfrak{m}_2$ is defined as $\mathfrak{m}_1 + \mathfrak{m}_2 = \mathfrak{m}_3$ with $\mathfrak{m}_1(a) + \mathfrak{m}_2(a) = \mathfrak{m}_3(a)$ for each $a \in \Sigma$. If $t \in \mathbb{N}$, then the *multiplication* $t \cdot \mathfrak{m}_1$, for a multiset $\mathfrak{m}_1$, is a multiset $\mathfrak{m}_2$ with $\mathfrak{m}_2(a) = t \cdot \mathfrak{m}_1(a)$ for each $a \in \Sigma$.

2.1.2 Directed graphs

A *directed graph* (*digraph*, for short) is a pair $G = (V, E)$ where V is the set of *nodes* and $E \subseteq V \times V$ is the set of *edges* of G. In the thesis, we assume that G is finite and connected. If G is acyclic, then G is called a *dag*. G is called a *directed rooted tree* if the following two conditions hold: (i) the underlying undirected graph of G is a *tree*, that is, an acyclic and connected graph, and (ii) there is a node $r \in V$, called the *root* of G, such that all the edges in G point towards r.

Let $p = (i_1, i_2, \ldots, i_l)$, $l \in [|V|]$, be a sequence of pairwise distinct vertices in V such that for every $j \in [l-1]$, $(i_j, i_{j+1}) \in E$. Then p is called a *directed path from* i_1 *to* i_l of G. If we do not want to emphasize the endpoints of p, then we say that p is a *directed path* of G. Moreover, if there is no danger of confusion, we will call directed, rooted trees and directed paths of G simply trees and paths, respectively. We will use the notation $i \rightsquigarrow_G j$ to denote that there exists a path from i to j in G. By writing that j *is reachable from* i *in* G we will mean that $i \rightsquigarrow_G j$.

2.1.3 Propositional logic

Denote Var the set $\{x_1, x_2, \ldots\}$ of *propositional variables* (*variables* for short) and let for a number $n \in \mathbb{N}$, $Var_n = \{x_1, \ldots, x_n\}$. *Propositional formulas* (or just *formulas* to be short) are well-formed terms built up in the usual way from variables and the following logical connectives: $\neg$ (*negation*), $\wedge$ (*conjunction*), $\vee$ (*disjunction*), and $\rightarrow$ (*implication*).

The variables and their negations are called *literals*. The variable occurring in a literal ℓ is called the *base* of ℓ. A literal ℓ is *positive* (resp. *negative*), if $\ell = x$ (resp. $\ell = \neg x$), where $x \in Var$.

A *clause* $\mathcal{C}$ is a formula of the form $\mathcal{C} = (\ell_1 \vee \ell_2 \vee \ldots \vee \ell_k)$, where $\ell_1, \ldots, \ell_k$, $k \in \mathbb{N}$, are literals with pairwise different bases. If $\mathcal{C} = x$ for some $x \in Var$, then $\mathcal{C}$ is a *positive unit clause*. A formula in *conjunctive normal form* (CNF for short) is a formula φ of the form $\varphi = \mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \ldots \wedge \mathcal{C}_m$, $m \in \mathbb{N}$, where for every $j \in [m]$, $\mathcal{C}_j$ is a clause.

A *truth assignment* of the variables is a function $\mathcal{I} : Var \rightarrow \{true, false\}$. Let $\mathcal{I}$ be a truth assignment. $\mathcal{I}$ can be extended to formulas in a natural way. A formula φ is *satisfiable* if $\mathcal{I}(\varphi) = true$, for some truth assignment $\mathcal{I}$. $\mathcal{I}(\varphi) = true$ will be often

denoted by $\mathcal{I} \models \varphi$. Notice that a clause is always satisfiable, since its literals have pairwise different bases. Let φ be a formula in CNF. Due to the commutativity of $\wedge$, we can view φ as a finite set of clauses. Let $\mathcal{I}$ be a truth assignment. It is easy to see that

$$\mathcal{I} \models \varphi \text{ if and only if for each clause } \mathcal{C} \in \varphi, \mathcal{I} \models \mathcal{C}.$$

2.1.4 Classical computational complexity classes

We begin the last part of the review with a rather tight complexity class. For this, we first define a quite restricted type of Turing machine. A *deterministic log-time Turing machine* [30] has a read-only *input tape* of length n (where n is the length of the input), a read-write *address tape* of length $O(\log n)$ and a constant number of read-write *working tapes* of length $O(\log n)$. This Turing machine is required to work in logarithmic time, therefore, the head of the input tape moves in a nonsequential manner: in each step, the machine has access to that position of the input tape whose number is written in binary on the address tape. If this number is greater than n, the machine reads the end-of-input symbol. Due to the time restrictions, this machine is able to read only $O(\log n)$ bits of the input, but still, it is capable of determining the length of its input, computing sums (and thus multiplying by constants), differences and logarithms of numbers, moreover, decoding simple pairing functions on strings of length $O(\log n)$ and extracting portions of the input of size $O(\log n)$ [30]. The class of functions computable by deterministic log-time Turing machines is denoted by $\mathbf{DLOGTIME}$. In the thesis, we will call deterministic log-time Turing machines $\mathbf{DLOGTIME}$ Turing machines.

Now, we continue with some well-known complexity classes. The class of problems solvable by deterministic (respectively, nondeterministic) Turing machines using *logarithmic space* is denoted by $\mathbf{L}$ (resp., $\mathbf{NL}$). Analogously, $\mathbf{P}$ (resp., $\mathbf{NP}$) denotes the class of problems solvable in *polynomial time* by deterministic (resp., nondeterministic) Turing machines. Moreover, the class of problems solvable by deterministic Turing machines using *polynomial space* is denoted by $\mathbf{PSPACE}$.

The class of decision problems $\mathbf{PP}$ is defined as follows. A language L is in $\mathbf{PP}$ if and only if there is a polynomial-time nondeterministic Turing machine M such that for every input u, $u \in L$ if and only if more than half of the possible computations of M on u end in the accepting state.

We finish the review with a class of counting problems, but first we introduce some necessary concepts. Let Σ be an alphabet and $Q \subseteq \Sigma^* \times \Sigma^*$ be a binary relation. Q is *polynomially decidable* if there exists a polynomial-time deterministic Turing machine that recognizes the language $\{\langle x, y\rangle : (x, y) \in Q\}$. Moreover, Q is *polynomially balanced* if there exists a $k \in \mathbb{N} - \{0\}$ such that for each $(x, y) \in Q$, $|y| \leq |x|^k$. That is, the length of y is polynomially bounded by the length of x. It is known that a decision

problem L is in $\mathbf{NP}$ if and only if there is a polynomially decidable and polynomially balanced binary relation Q such that

$$L = \{x \mid (x, y) \in Q\}. \tag{2.1}$$

Here, for a pair $(x, y) \in Q$, x is the *input* of L and y is a *certificate* of x.

In contrast to decision problems, for a counting problem, the question is not about its solvability but the number of its solutions. Formally, let $L \in \mathbf{NP}$ and let Q be the binary relation associated with L in Equality 2.1. The *counting problem* $\#L$ associated with L is defined as follows: given a word x, what is the number of words y satisfying $(x, y) \in Q$? In other words, for the counting problem $\#L$, the task is to count the number of certificates verifying an input x of L. Therefore, the solution of an instance of $\#L$ is a number in $\mathbb{N}$. Now, the class $\#\mathbf{P}$ is defined as follows:

$$\#\mathbf{P} = \{\#L \mid L \in \mathbf{NP}\}.$$

That is, $\#\mathbf{P}$ is the class of counting variants of problems in $\mathbf{NP}$.

2.2 Polarizationless P systems with active membranes

Now, we present a formal definition of polarizationless P systems with active membranes. Let O be an alphabet of objects and H be a finite set of membrane labels. We assume that H always contains the special label skin. Formally, a *membrane structure* is a triple (V, E, L) where (V, E) is a nonempty, finite, rooted, directed tree (with its edges directed towards the root) and $L : V \to H$ assigns *labels* to each node, such that only the root is labeled with the symbol skin, and it has to be labeled with skin. The nodes are called *membranes* of the structure. A membrane $x \in V$ with label h is called an *h-membrane*. We can assume that initially, each membrane has its unique label. A membrane that has only an outgoing edge is called an *elementary membrane*.

Before giving a formal definition of polarizationless P systems with active membranes, we first define membrane configurations based on the notion of membrane structures. A *membrane configuration* is a tuple (V, E, L, ω) where (V, E, L) is a membrane structure and $\omega : V \to \boldsymbol{\mathcal{M}}(O)$ is a function that assigns a finite multiset of objects to each membrane. A copy of an object a is called an *a-copy*. If x is a membrane and $a \in \omega(x)$ then we say that the *region of x contains an a-copy*, or just that *x contains an a-copy*. Let $C = (V, E, L, \omega)$ be a membrane configuration. The membrane configuration $C' = (V', E', L', \omega')$ is a *subconfiguration* of C if (V', E') is a subtree of (V, E), and L' and ω' are restrictions of L and ω to V', respectively.

Definition 2.2.1. A *polarizationless P system with active membranes* is a construct of the form $\Pi = (O, H, \mu, R)$, where O is the alphabet of objects, H is a finite set of

membrane labels, $\mu = (V, E, L, \omega)$ is the initial membrane configuration with L and ω mapping nodes of μ to H and O^*, respectively, and R is a finite set of *rules* of the following types:

(a) $[a \to v]_h$, for some $h \in H, a \in O, v \in O^*$ (*object evolution* rules);

(b) $a[\ ]_h \to [b]_h$, for some $h \in H$, $a, b \in O$ (*in-communication* rules);

(c) $[a]_h \to [\ ]_h b$, for some $h \in H$, $a, b \in O$ (*out-communication* rules);

(d) $[a]_h \to b$, for some $h \in H$, $a, b \in O$ (*membrane dissolution* rules);

(e) $[a]_h \to [b]_h[c]_h$, for some $h \in H$, $a, b, c \in O$ (*division* rules for elementary membranes).

The *left-hand side* of a rule is the string to the left of the symbol $\to$. We say that an a-copy uses a membrane x when a non-evolution rule, with a appearing on its left-hand side, is applied on x. The semantics of these rules will be described later, when we formally define a computation step of the P system. We note here that it will be defined in the same way as it was roughly defined in Chapter 1, when we discussed maximal parallelism. Nevertheless, we now define maximal parallelism precisely, in detail, because the thorough investigation of these P systems requires this explicitness.

Example 2.2.1. Let $\Pi_{ex} = (O_{ex}, H_{ex}, \mu_{ex}, R_{ex})$ be a polarizationless P system with active membranes and let $\mu_{ex} = (V_{ex}, E_{ex}, L_{ex}, \omega_{ex})$, where

- $O_{ex} = \{a, b\}$,
- $H_{ex} = \{h_1, \ldots, h_6, \mathsf{skin}\}$,
- $V_{ex} = \{env, s, x_1, \ldots, x_6\}$,
- $E_{ex} = \{(s, env), (x_1, s), (x_2, s), (x_3, x_1), (x_4, x_1), (x_5, x_2), (x_6, x_2)\}$,
- and the rules of R_{ex} are:

$$[a]_{h_1} \to [\ ]_{h_1} b, \qquad [b]_{h_1} \to a, \qquad [a]_{h_2} \to [\ ]_{h_2} a, \qquad [b]_{h_2} \to a,$$
$$a[\ ]_{h_3} \to [a]_{h_3}, \qquad a[\ ]_{h_4} \to [b]_{h_4}, \qquad [a]_{h_5} \to [b]_{h_5}[a]_{h_5}, \qquad [b \to aa]_{h_6}.$$

Furthermore, let $L_{ex} : E_{ex} \to H_{ex}$ be defined as follows:

$$L_{ex}(s) = \mathsf{skin}, \qquad L_{ex}(x_1) = h_1, \qquad L_{ex}(x_2) = h_2, \qquad L_{ex}(x_3) = h_3,$$
$$L_{ex}(x_4) = h_4, \qquad L_{ex}(x_5) = h_5, \qquad L_{ex}(x_6) = h_6.$$

Finally, we define $\omega_{ex} : V_{ex} \to O_{ex}^*$ as follows:

$$\omega_{ex}(env) = \varepsilon, \qquad \omega_{ex}(s) = \varepsilon, \qquad \omega_{ex}(x_1) = aab, \qquad \omega_{ex}(x_2) = abb,$$
$$\omega_{ex}(x_3) = ab, \qquad \omega_{ex}(x_4) = ab, \qquad \omega_{ex}(x_5) = aa, \qquad \omega_{ex}(x_6) = ab.$$

Figure 2.1 shows the membrane configuration μ_{ex}. The membranes x_3, x_4, x_5 and x_6 are elementary membranes.

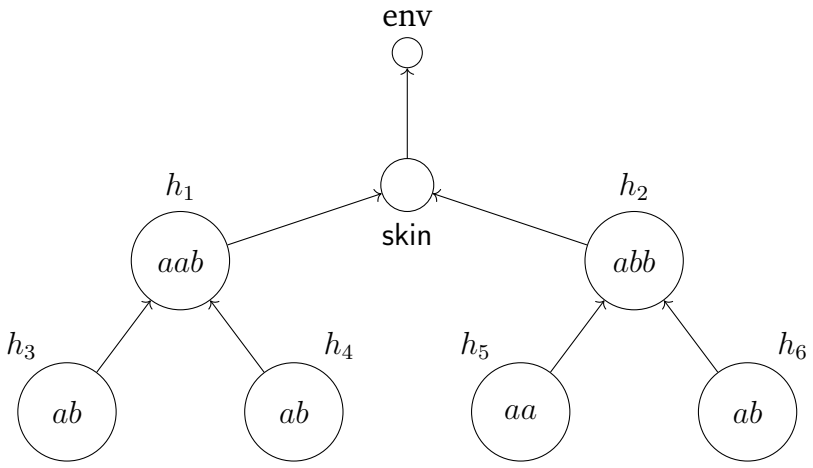


Figure 2.1: *The initial membrane configuration of* Π_{ex}

Next, we define a function on the edges of a configuration and call it *viable transition*. It is used to assign pairs of objects and rule types to the edges of the membrane configuration according to the notion of maximal parallelism.

Definition 2.2.2. Let $\Pi = (O, H, \mu, R)$ be a P system with active membranes and $C = (V, E, L, \omega)$ be a membrane configuration. The partial function $f : E \to O \times \{r_{\mathrm{b}}, r_{\mathrm{c}}, r_{\mathrm{d}}, r_{\mathrm{e}}\}$ is called a *viable transition* if it satisfies the following conditions (with $r_{\mathrm{b}}, r_{\mathrm{c}}, r_{\mathrm{d}}$ and r_{e} representing types of rules (b),(c),(d) and (e), respectively):

0. If $f(x, y) = (a, r)$ for some $x, y \in V, a \in O$ and $r \in \{r_{\mathrm{c}}, r_{\mathrm{d}}, r_{\mathrm{e}}\}$, then there has to be a rule with the left-hand side $[a]_{L(x)}$. Moreover, if x is not a leaf, then this rule cannot be a division rule. If $f(x, y) = (a, r_{\mathrm{b}})$, then there must be an in-communication rule with the left-hand side $a[\ ]_{L(x)}$.
1. In each membrane, there are at least as many copies of objects as leaving it. Formally, for each $x \in V$ and $a \in O$ we have

$$|\{(y, x) \in E : f(y, x) = (a, r_{\mathrm{b}})\}|$$
$$+|\{(x, y) \in E : r \in \{r_{\mathrm{c}}, r_{\mathrm{d}}, r_{\mathrm{e}}\}, f(x, y) = (a, r)\}| \leq \omega(x)(a).$$

   Note that the second set contains at most one edge, the one that leads from x to its parent.
2. For each membrane, if some copies of objects could use the membrane and cannot evolve inside it, but the membrane is not used, then all those copies are occupied with in-communication rules. Formally, for each edge $(x, y) \in E$ with

$f(x, y)$ being undefined, and for each $a \in O$ for which there exists a rule with the left-hand side $[a]_{L(x)}$, it has to be the case that

$$|\{(z, x) \in E : f(z, x) = (a, r_{\mathrm{b}})\}| = \omega(x)(a).$$

3. Similarly, for each membrane, if some copies of objects could use one of the inner membranes but it is not used, then each of those copies are occupied with evolution or using the membrane or using one of the inner membranes. Formally, for each edge (y, x) with $f(y, x)$ being undefined, and for each $a \in O$ for which there exists a rule with the left-hand side $a[\ ]_{L(y,x)}$, it has to be the case that

$$\begin{aligned} &|\{(y', x) \in E : f(y', x) = (a, r_{\mathrm{b}})\}| \\ +&|\{(x, z) \in E : r \in \{r_{\mathrm{c}}, r_{\mathrm{d}}, r_{\mathrm{e}}\}, f(x, z) = (a, r)\}| = \omega(x)(a) - k, \end{aligned}$$

where k is the number of a-copies that are chosen to evolve inside x according to an evolution rule $[a \to v]_{L(x)}$. Again, observe that the second term of the sum is either zero or one.

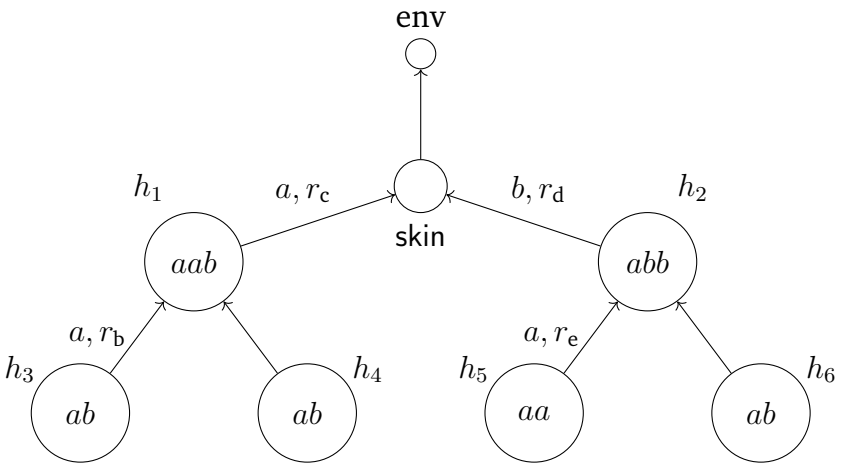


Figure 2.2: *A viable transition for the configuration of Example 2.2.1. The selected rules for application are* $[a]_{h_1} \to [\ ]_{h_1} b$*,* $[b]_{h_2} \to a$*,* $a[\ ]_{h_3} \to [a]_{h_3}$*,* $[a]_{h_5} \to [b]_{h_5}[a]_{h_5}$ *and* $[b \to aa]_{h_6}$*.*

Notice that viable transitions do not define explicitly which rules are attached to the membranes.

A viable transition of the P system Π_{ex} of Example 2.2.1 can be seen in Figure 2.2.

Definition 2.2.3. Let $C = (V, E, L, \omega)$ be a membrane configuration and $f : E \to O \times \{r_{\mathrm{b}}, r_{\mathrm{c}}, r_{\mathrm{d}}, r_{\mathrm{e}}\}$ be a viable transition for C. Then the computation step $C \overset{f}{\Rightarrow} C'$

for the membrane configuration C' is defined via several substeps discussed below. During the definition, we demonstrate the given steps by an example. We use the membrane system given in Example 2.2.1 and show the application of its rules on the configuration shown in Figure 2.1 according to the viable transition given in Figure 2.2. The example is given via the corresponding figures below.

1. (Membrane attachments.) First, we remove those copies of objects from the membranes that use them. The new configuration $C_1 = (V, E, L, \omega_1)$ is defined as follows: for each membrane $x \in V$ and object $a \in O$, let

$$\begin{aligned}\omega_1(x)(a) = \omega(x)(a) &- |\{(y, x) \in E : f(y, x) = (a, r_{\mathrm{b}})\}| \\ &- |\{(x, y) \in E : r \in \{r_{\mathrm{c}}, r_{\mathrm{d}}, r_{\mathrm{e}}\}, f(y, x) = (a, r)\}|.\end{aligned}$$

   Since f is viable for C, these values are nonnegative for each x and a.

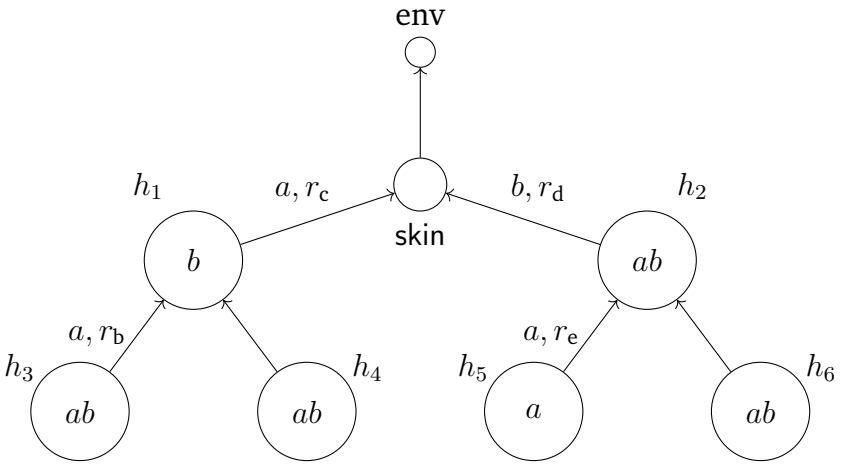


Figure 2.3: *After membrane attachment. Two a-copies, a b-copy and an a-copy are removed from the h_1-membrane, the h_2-membrane and the h_5-membrane, respectively.*

2. (Evolutions.) Next, we apply the evolution rules. The new configuration $C_2 = (V, E, L, \omega_2)$ is defined as follows: for each membrane $x \in V$, let $\omega_1(x) = a_1 a_2 \dots a_m$ and $h = L(x)$. Then, define $\omega_2(x)$ as $u_1 u_2 \dots u_m$ where

$$u_i = \begin{cases} u & \text{if there is an evolution rule } [a_i \to u]_h \\ a_i & \text{otherwise.} \end{cases}$$

3. (Communications.) Applying the in- and out-communication rules we get $C_3 = (V, E, L, \omega_3)$ which is defined as follows: for each membrane $x \in V$ and $b \in O$,

let

$$\omega_3(x)(b) = \omega_2(x)(b) + |\{(y,x) \in E : f(y,x) = (a, r_{\mathrm{c}}), [a]_{L(y)} \to [\ ]_{L(y)} b \in R\}| \\ + |\{(x,y) \in E : f(x,y) = (a, r_{\mathrm{b}}), a[\ ]_{L(x)} \to [b]_{L(x)} \in R\}|.$$

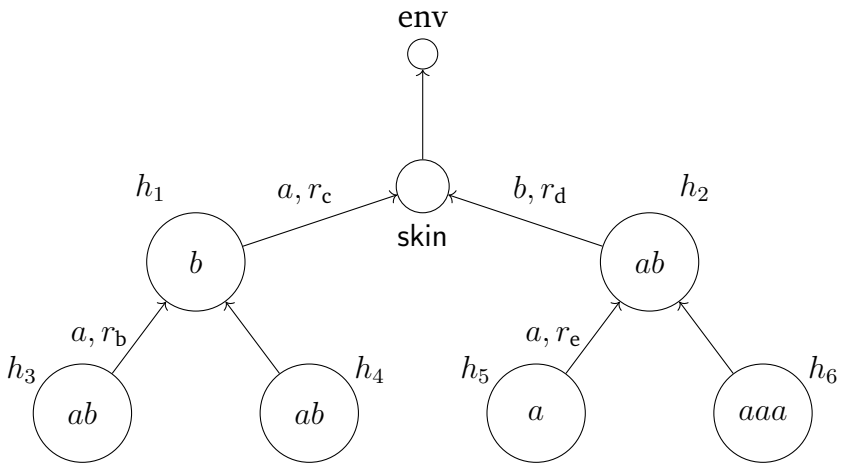


Figure 2.4: *After evolutions. Inside the h_6-membrane the b-copy got rewritten to aa.*

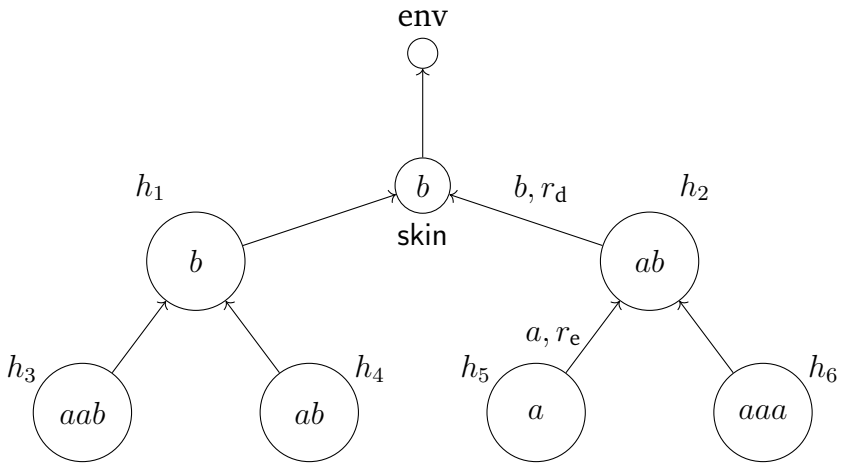


Figure 2.5: *After communications. The h_1-membrane released a as b upwards, the h_3-membrane released a as a downwards.*

4. (Dissolutions.) Applying the dissolution rules, the current configuration is $C_4 = (V_4, E_4, L_4, \omega_4)$ which we define as follows. Initially let $C_4 = C_3$. Iterating from the leaves towards the root, we dissolve membranes step by step as follows: if for an edge $(x,y) \in E_4$, $f(x,y) = (a, r_{\mathrm{d}})$ so that $[a]_{L_4(x)} \to b \in R$ for some $a, b \in O$, then we set $\omega_4(y) = \omega_4(y) + \omega_4(x) + b$. Moreover, for each $z \in V_4$ with $(z,x) \in E_4$, we add a new edge (z,y) to E_4, set $E_4 = E_4 - \{(z,x)\}$. Furthermore,

we set $E_4 = E_4 - \{(x, y)\}$ and $V_4 = V_4 - \{x\}$. We repeat this process till we handled all the dissolution-marked membranes.

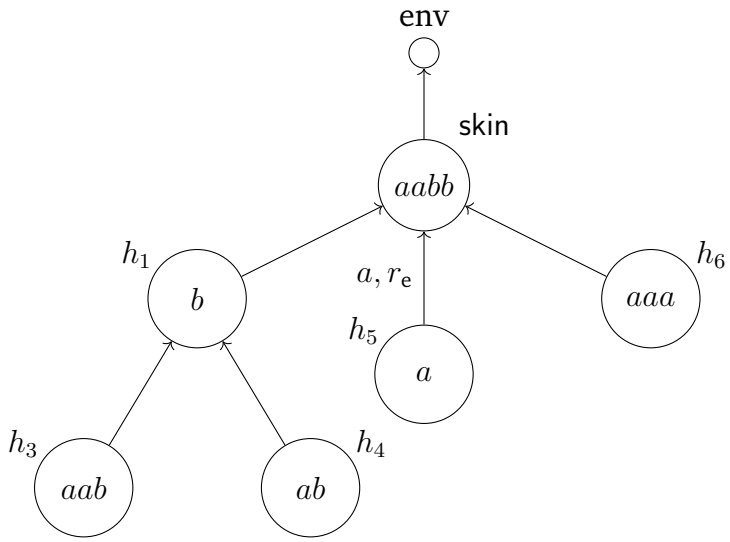


Figure 2.6: *After dissolutions. The h_2-membrane got dissolved under the rule $[b]_{h_2} \to a$.*

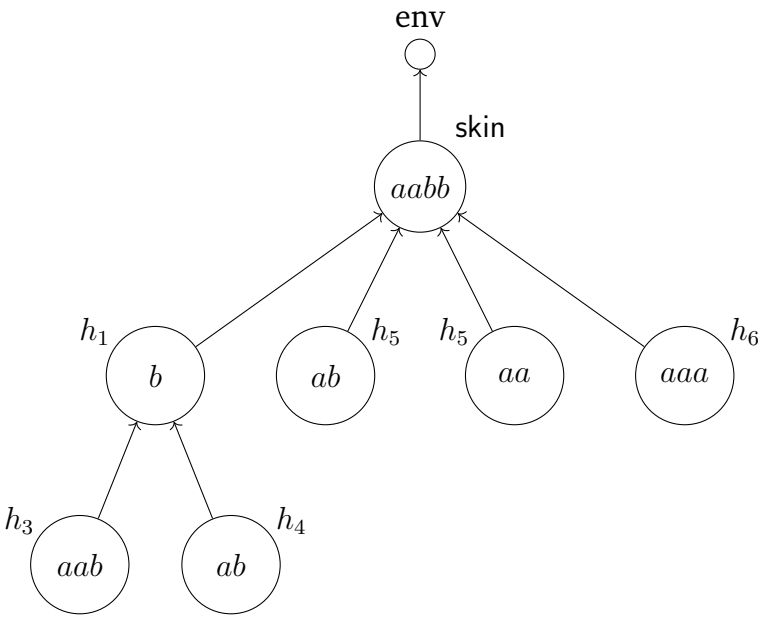


Figure 2.7: *After divisions. The h_5-membrane became divided under the rule $[a]_{h_5} \to [b]_{h_5}[a]_{h_5}$.*

5. (Divisions.) Finally, $C' = (V', E', L', \omega')$ is defined as follows. Initially let $C' = C_4$. For each edge $(x, y) \in E'$ such that $f(x, y) = (a, r_e)$ and there is a division rule $[a]_{L'(x)} \to [b]_{L'(x)}[c]_{L'(x)}$ for some $a, b, c \in O$, let $V' = V' \cup \{x'\}$, where x' is

a new child of y with $\omega'(x') = \omega'(x) + c$ and $L'(x', y) = L'(x, y)$. Then we set $\omega'(x) = \omega'(x) + b$.

The membrane configuration we end in up after this last step is C' with $C \overset{f}{\Rightarrow} C'$. If there is such a viable f, then we write $C \Rightarrow C'$.

2.3 Uniform families of recognizer P systems

Recognizer P systems [48] were introduced to provide a framework in which P systems can be considered as *deciders* of decision problems.

Definition 2.3.1. A P system Π is a *recognizer P system* if it satisfies the following properties:

1. Π has a designated *input membrane* and a designated *output membrane*,
2. the alphabet of objects has two designated elements yes and no,
3. all computations of Π halt, and
4. each computation of Π must produce in the output membrane exactly one yes or no (but not both), and these objects are produced exactly in the last step of the computation.

Recognizer P systems, being nondeterministic computation models, can have different computations with different outputs on the same input. However, to use them as deciders, confluent recognizer P systems are usually considered.

Decision problems in membrane computing are usually solved by *uniform* families of recognizer P systems [43]. First, we present a general definition of uniformity by considering classes of functions which will be useful in Chapter 4 when defining **DLOGTIME**-uniformity.

Definition 2.3.2. Let $\mathcal{L}$ be a class of functions and D be a decision problem. A family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ of P systems is *$\mathcal{L}$-uniform* if there exists a function $f \in \mathcal{L}$ such that $f(1^n)$ is a reasonable encoding of $\Pi(n)$. That is, f maps the unary representation of n to an encoding of the P system processing all instances of D with size n. Moreover, we say that an $\mathcal{L}$-uniform family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ of recognizer P systems *solves D in polynomial time* if the following conditions hold:

- for each $n \in \mathbb{N}$, the initial membrane structure of $\Pi(n)$ has a designated membrane called the *input membrane*,
- there is a function $cod \in \mathcal{L}$ mapping each instance s of D into a multiset $cod(s)$ over the objects of $\Pi(n)$, where n is the size of s, and
- there is $k \in \mathbb{N}$ such that for each instance s of D with size n, starting $\Pi(n)$ with $cod(s)$ in its input membrane,

- each computation of $\Pi(n)$ halts in at most n^k steps and
- all computations of $\Pi(n)$ produce yes if s is a positive instance of D and all computations produce no otherwise.

If we want to decide a problem D with an $\mathcal{L}$-uniform family of recognizer P systems, then it is natural to assume that the devices computing functions f and cod are not capable of solving problem D. Throughout the thesis, when we say that a family of P systems solves a problem *efficiently*, we mean that it solves the problem in polynomial time.

Since we will use the concepts of **L**-*uniformity* and **P**-*uniformity* several times in the thesis, we now define them specifically.

Definition 2.3.3. A family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ of P systems is **L**-*uniform* (respectively, **P**-*uniform*) if there is a deterministic Turing machine using logarithmic space (resp., working in polynomial time) that computes $\Pi(n)$ whenever it is started with 1^n, that is, with the unary representation of $n \in \mathbb{N}$ on its input tape. Furthermore, let D be a decision problem and $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ be an **L**-uniform (resp., a **P**-uniform) family of recognizer P systems. We say that $\mathbf{\Pi}$ *solves* D *in polynomial time* if the following conditions hold.

- There is a function cod mapping instances of D into multisets of objects such that cod is computable by a deterministic Turing machine using logarithmic space (resp., working in polynomial time), and
- there is $k \in \mathbb{N}$ such that for every instance x of D with size n, starting $\Pi(n)$ with $cod(x)$ in its input membrane, each computation of $\Pi(n)$ halts in at most n^k steps and produces yes in the output membrane if and only if x is a positive instance of D.

Chapter 3

On the power of membrane dissolution

In this chapter, we consider polarizationless P systems with active membranes using only dissolution rules and investigate their computational power. As a result, we show that an **NL**-complete variant of the REACHABILITY problem can be solved in polynomial time by these P systems under a suitable uniformity condition.

3.1 Introduction

Removing dissolution rules significantly affects the computational power of polarizationless P systems with active membranes. For example, in the previously recommended work [19], it was shown that these P systems can solve only problems in **P** even if non-elementary membrane division is allowed. This result was strengthened in [35] where it was shown that without polarizations these P systems can solve only rather trivial problems under tight uniformity conditions. For a recent survey on exploring the boundary between **P** and **NP** in terms of membrane computing, we refer again to [36].

The discussion above suggests that dissolution rules have an important role in polarizationless P systems with active membranes. This motivates the idea of studying what polarizationless P systems with active membranes are capable of regarding their computational power, when only dissolution rules are allowed to be applied.

In membrane computing, it is common to use polynomial-time uniform families of recognizer P systems to solve decision problems [48]. Such a family of P systems can solve a problem A in **P** in a trivial way, as the answer for an input I of A is computable already during the polynomial-time encoding of I. This was exploited in [19] to give a characterization of the complexity class **P** by polarizationless P systems with active membranes using no dissolution rules. In [32], the computational power of **L**-uniform families of these P systems was investigated. Since deterministic logarithmic-

space Turing machines are presumably weaker than deterministic polynomial-time Turing machines, the uniformity condition used in [32] is presumably tighter than the polynomial-time uniformity used in [19]. In [32] it turned out that **L**-uniform families of polarizationless P systems with active membranes using no dissolution rules can solve only problems in **NL**. However, in [32] it remained open whether these P systems can solve all the problems in **NL**.

In [35], an even tighter uniformity condition, the $\mathbf{FAC}^0$-uniformity was considered. $\mathbf{FAC}^0$ is the class of functions computable by **DLOGTIME**-uniform constant-depth polynomial-size Boolean circuits with unbounded fan-in. In this case, both the encoding of the P systems in the family and the encoding of the input are functions in $\mathbf{FAC}^0$. In [35], it turned out that $\mathbf{FAC}^0$-uniform recognizer families of polarizationless P systems with active membranes using no dissolution rules can solve efficiently only problems in $\mathbf{AC}^0$, which is a rather tight complexity class. Analogously, $\mathbf{AC}^0$ is the class of decision problems solvable by constant-depth polynomial-size **DLOGTIME**-uniform Boolean circuits with unbounded fan-in. It is known to be strictly included in **L**, they cannot even solve the PARITY problem (deciding whether a binary string contains an odd number of 1s). That is, these P systems cannot solve problems other than those computable by the circuits used to define the uniformity condition. Consequently, the computational power of these P systems is rather limited. This initiated a research line where the lower bound on the computational power of restricted variants of polarizationless P systems was investigated using reasonable tight uniformity conditions (see, e.g. [15, 17, 29, 33, 34]).

In this chapter, we show that polarizationless P systems using only dissolution rules can solve **NL**-complete problems in polynomial time. In fact, we will solve an **NL**-complete variant of the REACHABILITY problem with these P systems. In the REACHABILITY problem, one has to decide whether there is a path between two distinguished nodes in a directed graph. Consider a directed graph G with a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$. In our P systems, an edge $(i, j) \in \mathcal{E}$, $i, j \in \mathcal{V}$ is encoded by an object $e_{i,j}$. Clearly, if (i, j) and (j, k) are edges of G, then there is a path from i to k in G. Accordingly, our P systems implement, roughly, instructions of the following form:

if $e_{i,j}$ and $e_{j,k}$ occur in the membrane structure **then introduce** $e_{i,k}$.

These instructions will be implemented by the appropriate use of dissolution rules in a linearly nested membrane structure. The P systems need to check whether objects $e_{i,j}$ and $e_{j,k}$ occur together in a region of the P system or not such that it continues the computation even if one of the objects $e_{i,j}$ or $e_{j,k}$ is not present in this membrane region. In general, to decide whether two objects occur in the same region requires a certain amount of context-sensitivity implemented in the P systems. This context-sensitivity was implemented, for example, in [15] with the help of the membrane

polarizations. In [54], the **NP**-complete SAT problem was solved by polarizationless P systems with active membranes using no dissolution rules. Here, however, evolution rules with minimal cooperation, that is, evolution rules with two objects on the left-hand side were used to implement the required context-sensitivity (see also [37]). To give an intuition of how our P systems solve this task, we give a (rather simplified) example.

Assume that we have a membrane structure $[[[\,]_2]_1]_s$ where $[\,]_2$ contains an object b and it may contain an object a as well. We want to set up a P system to solve the following task: if $[\,]_2$ contains an a, then b gets to membrane $[\,]_s$ so that meanwhile it evolves to b'. If, on the other hand, a does not occur in $[\,]_2$, then we want b to appear in $[\,]_s$ without any evolution. To check whether both a and b occur in $[\,]_2$, it is enough to add rules $[a]_2 \to \#$ and $[b]_1 \to b'$, where $\#$ is a dummy object not used elsewhere. However, the membrane structure described above cannot handle the case when a is not present. Therefore, we extend this membrane structure as follows:

$$\Big[\Big[\,\big[\,[\,]_l\big]_l\,\Big]_2\,\big[[\,]_l\big]_l\Big]_1\Big]_s.$$

Let us put an object c into the innermost membranes with label l and an object b in the membrane with label 2. Then, we get the following membrane configuration:

$$\Big[\Big[\,\big[b\,[[c]_l\,]_l\,\big]_2\,[[c]_l\,]_l\Big]_1\Big]_s.$$

Let us, moreover, add the following rules to the system: $[c]_l \to c, [c]_2 \to \#, [c]_1 \to \#$. Notice that the P system still has the rules $[a]_2 \to \#$ and $[b]_1 \to b'$. Then, the P system has the following computation:

$$\Big[\Big[\,\big[b\,[[c]_l\,]_l\,\big]_2\,[[c]_l\,]_l\Big]_1\Big]_s \Rightarrow \Big[\Big[\,\big[b\,[c]_l\,\big]_2\,[c]_l\,\Big]_1\Big]_s \Rightarrow \Big[\Big[\,\big[b\,c\,\big]_2\,c\,\Big]_1\Big]_s \Rightarrow [\,b\,\#\,\#\,]_s.$$

That is, b reached the membrane with label s without evolving to b' as we desired. Notice that b is never present in membrane 1. Therefore, it cannot trigger the rule $[b]_1 \to b'$. Let us see now how the P system works if a occurs besides b in membrane 2 at the beginning of the computation:

$$\Big[\Big[\,\big[a\,b\,[[c]_l]_l\,\big]_2\,[[c]_l]_l\,\Big]_1\Big]_s \Rightarrow \Big[\big[\,\#\,b\,[c]_l\;[c]_l\,\big]_1\Big]_s \Rightarrow [\,\#\,b'\,c\;c\,]_s.$$

That is, at the end of the computation, b' occurs in the membrane with label s as we desired. The key in this construction is the use of linearly nested membranes and dissolution rules to handle the case when some objects do not appear in the membrane structure.

Extending the above-described idea appropriately, we can set up our P systems so that they can solve the REACHABILITY problem in polynomial time using only dissolution rules. Notice that the REACHABILITY problem was already solved by P systems (see, e.g. [14, 32, 33]) but in these solutions, the authors used other types of rules instead of dissolution rules.

The rest of the chapter is organized as follows. In the next section, we introduce dissolution P systems, moreover, we define the REACHABILITY problem and Algorithm 1 that efficiently solves it. Then, in Section 3.3, we present an **L**-uniform family of recognizer P systems with active membranes using only dissolution rules that solves REACHABILITY in polynomial time by implementing Algorithm 1. Finally, in Section 3.4, we make some concluding remarks.

3.2 Preliminaries

A *dissolution P system* is a recognizer polarizationless P system with active membranes that employs only dissolution rules. More formally, a dissolution P system is a construct of the form $\Pi = (O, H, \mu, R)$, where O is the alphabet of objects, H is a finite set of membrane labels, $\mu = (V, E, L, \omega)$ is the initial membrane configuration with L and ω mapping nodes of μ to H and O^*, respectively, and R is a finite set of rules of the form

$$[a]_h \to b,$$

where $h \in H$ and $a, b \in O$.

In this chapter, we consider the following variant of the well-known **NL**-complete REACHABILITY problem (for the basic notations of graph theory we use in this chapter, we refer to Chapter 2). Consider a digraph $G = (\mathcal{V}, \mathcal{E})$. We call G *topologically sorted*, if $\mathcal{V} = \{1, 2, \ldots, n\}$, $n \geq 2$, and for every $(i, j) \in \mathcal{E}$, $i < j$. Then REACHABILITY is defined as follows:

> **Input**: a topologically sorted digraph $G = (\mathcal{V}, \mathcal{E})$.
> **Output**: yes if $1 \rightsquigarrow_G n$ and no otherwise.

It is known that REACHABILITY defined in this restricted form is still **NL**-complete [22]. We will show that REACHABILITY can be solved in polynomial time by an **L**-uniform family of dissolution P systems. In fact, our P systems will implement a function $reachability$ given in Algorithm 1. This algorithm is based on the following observation. Let $G = (\mathcal{V}, \mathcal{E})$ be a topologically sorted digraph with n nodes. Moreover, let $i \in [2, n-1]$ and $j \in [i+1, n]$. Clearly, if a node i is reachable from 1 in G and $(i, j) \in \mathcal{E}$, then node j is reachable from 1 in G as well. Thus, the algorithm, roughly, checks whether $1 \rightsquigarrow_G i$ and $(i, j) \in \mathcal{E}$. If so, then it extends $\mathcal{E}$ with the edge $(1, j)$. This extension is executed by the corresponding cycle of the innermost loop

(Lines 6-9) which we call the (i,j)*th edge operation* of Algorithm 1. Notice that the number of these edge operations is $(n-2)+(n-3)+\ldots+2+1=\frac{n^2-3n+2}{2}$.

Algorithm 1 Decision of REACHABILITY

```
1: function reachability(G)
2:     E_1 = E
3:     for i = 2 to n − 1 do
4:         Ê_i = E_{i−1}
5:         for j = i + 1 to n do
6:             if (1, i) ∈ Ê_i and (i, j) ∈ Ê_i then
7:                 Remove(Ê_i,(i, j))
8:                 Add(Ê_i,(1, j))
9:             end if
10:        end for
11:        E_i = Ê_i
12:    end for
13:    if (1, n) ∈ E_{n−1} then
14:        return true
15:    else
16:        return false
17:    end if
18: end function
```

Although it is intuitive, we show that Algorithm 1 is correct and complete.

Proposition 1. *Let $G=(\mathcal{V},\mathcal{E})$ be a topologically sorted digraph with n nodes. Then, $1 \rightsquigarrow_G n$ if and only if $(1,n)\in\mathcal{E}_{n-1}$ in Algorithm 1.*

Proof. To prove this statement, we show a more general statement:

$$\text{for every } j\in[2,n],(1,j)\in\mathcal{E}_{j-1} \text{ if and only if } 1\rightsquigarrow_G j. \tag{3.1}$$

We show Statement (3.1) by induction on j. If $j=2$, then we need to show that $(1,2)\in\mathcal{E}_1$ if and only if $1\rightsquigarrow_G 2$. $(1,2)\in\mathcal{E}_1$ clearly implies $1\rightsquigarrow_G 2$. If $1\rightsquigarrow_G 2$, then, using that G is topologically ordered, 2 can only be reached from 1 in one step. Consequently, $(1,2)\in\mathcal{E}_1$. Now, assume that Statement (3.1) holds when $j\in[2,n-1]$. We show it for $j+1$.

First, assume that $(1,j+1)\in\mathcal{E}_j$. Then, we have two cases. If $(1,j+1)\in\mathcal{E}_1$, then $1\rightsquigarrow_G j+1$ clearly holds. If $(1,j+1)\notin\mathcal{E}_1$, then there is a number $i\in[2,j]$ such that $(1,j+1)$ is added to $\hat{\mathcal{E}}_i$ during the $(i,j+1)$th edge operation. Thus, both $(1,i)$ and $(i,j+1)$ are in $\mathcal{E}_{i-1}$. From $(1,i)\in\mathcal{E}_{i-1}$ we get $1\rightsquigarrow_G i$ by the induction hypothesis. From $(i,j+1)\in\mathcal{E}_{i-1}$, using the fact that edges with a source node greater than 1 are never introduced by edge operations, we get $(i,j+1)$ in $\mathcal{E}_1$. Consequently, $1\rightsquigarrow_G j+1$.

Now, assume that $1 \rightsquigarrow_G j+1$. If $(1, j+1) \in \mathcal{E}_1$, then $(1, j+1) \in \mathcal{E}_j$, since the edges with the source node 1 are not removed by the edge operations. If $(1, j+1) \notin \mathcal{E}_1$, then there is $i \in [2, j]$ such that $1 \rightsquigarrow_G i$ and $(i, j+1) \in \mathcal{E}_1$. From $1 \rightsquigarrow_G i$ we get $(1, i) \in \mathcal{E}_{i-1}$ by the induction hypothesis. $(i, j+1)$ can be removed only by the $(i, j+1)$th edge operation, and it is removed by it since $(1, i) \in \hat{\mathcal{E}}_i$. However, after removing $(i, j+1)$, $(1, j+1)$ is introduced in $\hat{\mathcal{E}}_i$ by the $(i, j+1)$th edge operation. Using again the fact that the edges with the source node 1 are not removed during any edge operation, we get $(1, j+1) \in \mathcal{E}_j$. □

Note that Algorithm 1 is not the most efficient one concerning the number of steps. For example, it would remain complete and correct even if Line 7 was removed. Moreover, it is easy to see that if there is no path from 1 to i, then it is superfluous to run the inner loop. Nevertheless, this is an algorithm that is suitable for dissolution P systems to implement.

3.3 Solving REACHABILITY with dissolution P systems

In this section, we prove the following main result of the chapter.

Theorem 1. *The* REACHABILITY *problem can be solved in polynomial time by an* **L***-uniform family of dissolution P systems.*

The rest of this section is devoted to the proof of this theorem. We give an **L**-uniform family $\mathbf{\Pi} = \{\Pi(n) \mid n \geq 2\}$ of deterministic dissolution P systems where $\Pi(n)$ is devoted to decide REACHABILITY for topologically sorted graphs of n nodes by simulating Algorithm 1.

For the remainder of this section, let us fix a topologically sorted graph $G = (\mathcal{V}, \mathcal{E})$ with n nodes and m edges. First, we define the encoding of G, denoted by $cod(G)$, as a subset of $\Sigma(n) = \{e_{i,j} \mid i, j \in [n]\}$ in the following way:

$$cod(G) = \{e_{i,j} \mid (i, j) \in \mathcal{E}\}.$$

3.3.1 The construction of $\Pi(n)$

The P system $\Pi(n) = (O, H, \mu, R)$ is defined as follows:

- $O = \Sigma(n) \cup \{c, \#, yes, no\}$,
- $H = \{h_{i,j}, h'_{i,j} \mid i \in [2, n-1], j \in [i+1, n]\} \cup \{\mathsf{skin}, \mathsf{halt}, \mathsf{input}, l, l_c, g\}$,
- the labels of the input and output membranes are input and skin, respectively, and
- the initial membrane configuration (μ) and the set of rules (R) are defined below.

The initial membrane configuration. We define $\mu = (V, E, L, \omega)$ with the help of Figure 3.1 as follows. The rectangles and circles denote subconfigurations and membranes of μ, respectively. An edge going out of a rectangle represents an edge going out of the root of the subconfiguration represented by that rectangle. The name of a subconfiguration is placed beside the corresponding rectangle. Likewise, the label $L(x)$ of a membrane x is placed beside x. Moreover, the object multiset $\omega(x)$ associated initially to a membrane x is written in the corresponding circle. Empty circles represent empty multisets. The root of the membrane structure is the skin-membrane, which has an only child with label halt. Initially, both the skin-membrane and the halt-membrane are empty.

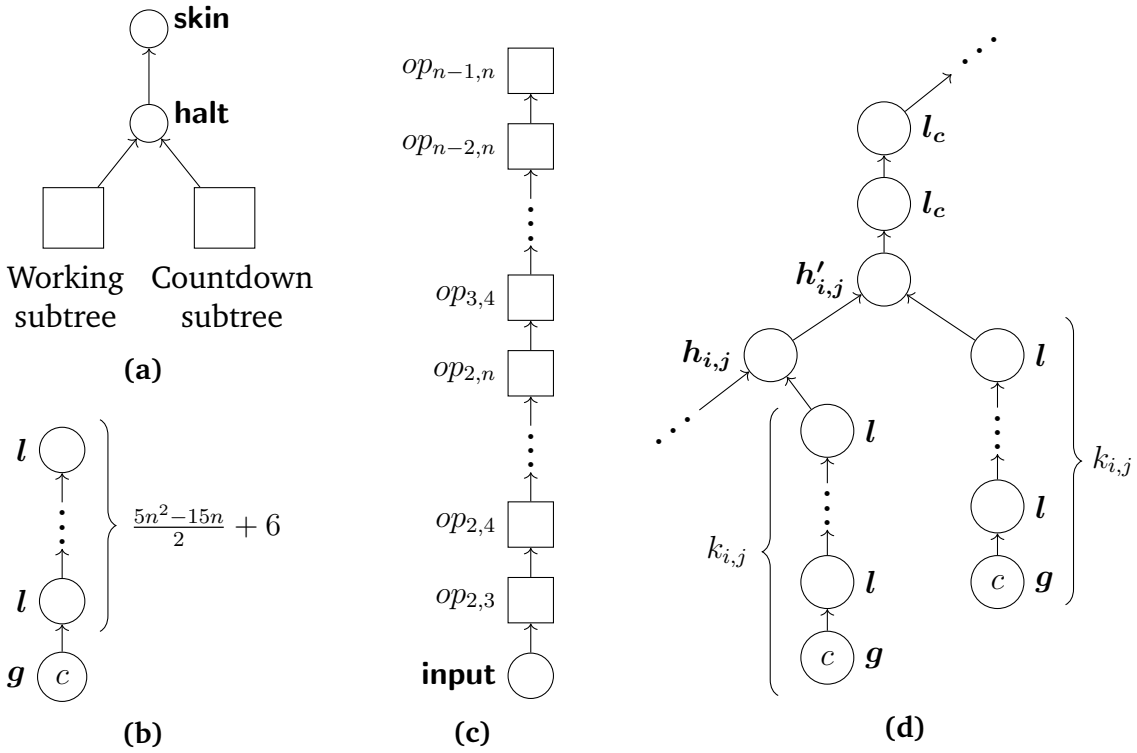


Figure 3.1: *The initial membrane configuration (a), the countdown subtree (b), the working subtree (c), and the operational subconfiguration $op_{i,j}$ (d). The rectangles denote subconfigurations, the circles denote membranes.*

Below the halt-membrane there are two subconfigurations called the *working subtree* and the *countdown subtree*, respectively (see Figure 3.1a). The working subtree (Figure 3.1c) will be used to decide whether node n is reachable from node 1. More precisely, this part will produce the object $e_{1,n}$ in the halt-membrane if and only if there is a path from 1 to n. Then $e_{1,n}$ triggers the dissolution of the halt-membrane, resulting yes in the skin. The countdown subtree (Figure 3.1b) will be used to pro-

duce a c-copy in the halt-membrane if and only if there is no path from 1 to n, that is, the object $e_{1,n}$ was not produced in the halt-membrane by the working subtree. Then this c-copy triggers the dissolution of halt, resulting no in the skin. The countdown subtree has $\frac{5n^2-15n}{2}+6$ linearly nested l-membranes and an elementary g-membrane in the innermost l-membrane. Initially, all l-membranes are empty, and for the only g-membrane x, $\omega(x) = c$. Notice that the number of the l-membranes is $5 \cdot \eta + 1$, where η is the number of edge operations in Algorithm 1.

The working subtree is divided into several parts (see Figure 3.1c). A subconfiguration $op_{i,j}$, $i \in [2, n-1]$, $j \in [i+1, n]$, is called an *operational subconfiguration* and corresponds to the (i,j)th edge operation of Algorithm 1. For the sake of simplicity, in the rest of the chapter, all the subconfigurations yielded from the operational subconfiguration $op_{i,j}$ during the computation of $\Pi(n)$ will be referred to as $op_{i,j}$, too.

Figure 3.1d shows the operational subconfiguration $op_{i,j}$ of μ. The two outmost membranes in $op_{i,j}$ are two l_c-membranes. The inner l_c-membrane contains an $h'_{i,j}$-membrane which contains an $h_{i,j}$-membrane. If $j > 3$, then the $h_{i,j}$-membrane contains the next operational subconfiguration below $op_{i,j}$ according to Figure 3.1c. If $j = 3$ (and thus $i = 2$), then $h_{i,j}$ contains the input membrane of $\Pi(n)$ with label input. For the input-membrane x, $\omega(x) = c$.

Moreover, for every $i \in [2, n-1]$ and $j \in [i+1, n]$, the $h_{i,j}$-membrane and also the $h'_{i,j}$-membrane contain a membrane structure of linearly nested l-membranes. The number $k_{i,j}$ of the l-membranes in each membrane structure is defined recursively as follows:

$$k_{i,j} = \begin{cases} 2, \text{ if } i = 2, j = 3, \\ 5 + k_{i,j-1}, \text{ if } i \in [2, n-1], j \in [i+2, n], \\ 5 + k_{i-1,n}, \text{ if } i \in [3, n-1], j = i+1. \end{cases} \tag{3.2}$$

That is, when $i > 2$ or $j > i+1$, the $h_{i,j}$-membrane (resp. the $h'_{i,j}$-membrane) contains five more l-membranes than the number of these membranes in the corresponding membrane of the operational subconfiguration right below $op_{i,j}$.

Furthermore, both innermost l-membranes in $op_{i,j}$ contain an elementary g-membrane initially containing a c-copy. For any other membrane x in $op_{i,j}$ with $L(x) \neq g$, $\omega(x) = \varepsilon$. Consider an operational subconfiguration $op_{i,j}$, for some $i \in [2, n-1]$ and $j \in [i+1, n]$. For simplicity, the $h_{i,j}$-membrane and $h'_{i,j}$-membrane of $op_{i,j}$ is often called the *h-membrane* and the *h'-membrane* of $op_{i,j}$, respectively. Moreover, when we say that an operational subconfiguration op' is *right above* $op_{i,j}$, we mean that $op' = op_{i',j'}$ where $i' = i$, $j' = j+1$ if $j < n$ and $i' = i+1$, $j' = i+2$ otherwise. Furthermore, by saying that $\Pi(n)$ *dissolves* $op_{i,j}$, we mean that $\Pi(n)$ dissolves all the membranes initially placed in $op_{i,j}$.

With this, we have finished the description of the initial membrane configuration μ.

The rules of $\Pi(n)$. We define the set R of rules as follows:

- $[c]_{\mathsf{input}} \to \#$,
- $[c]_g \to c$,

- $[c]_l \to c$,
- $[e_{1,i}]_{h_{i,j}} \to e_{1,i}$,
- $[e_{i,j}]_{h'_{i,j}} \to e_{1,j}$,
- $[c]_{h_{i,j}} \to c$,
- $[c]_{h'_{i,j}} \to c$,
- $[c]_{l_c} \to \#$,

$\Big\}\ i \in [2, n-1], j \in [i+1, n]$

- $[e_{1,n}]_{\mathsf{halt}} \to yes$,
- $[c]_{\mathsf{halt}} \to no$.

3.3.2 An overview of the computation

Assume that $\Pi(n)$ is started with the multiset $cod(G)$ in the input-membrane. Recall that G is a topologically sorted graph with n nodes and m edges and that $cod(G)$ is the encoding of G. We describe the computation of $\Pi(n)$ and along with it, we also show the correctness of $\Pi(n)$.

During the computation, the input objects in $cod(G)$ may evolve to objects in $\Sigma(n)$ by means of dissolution rules. Nevertheless, the number of these objects remains m during the whole computation (it may be, however, that an object occurs multiple times in a configuration). Moreover, in each configuration of the computation, these m objects always occur in the same region as they never get separated due to the nature of dissolution rules. We call the multiset that contains the objects derived from $cod(G)$ the *edge-multiset*.

In the first step of the computation, $\Pi(n)$ uses the following rules:

- $[c]_{\mathsf{input}} \to \#$,
- $[c]_g \to c$.

Since the input-membrane initially contains a c-copy, it gets dissolved by the application of the first rule above. After applying this rule, the c-copy becomes $\#$, a dummy object that does not appear on the left-hand side of any rule, so it will not be used anymore. For that very reason, whenever a $\#$-copy occurs in a membrane, we do not consider it to be part of the content of the region. Therefore, we can say that after the first step, the $h_{2,3}$-membrane in $op_{2,3}$ contains only the objects of

$cod(G)$. Still in the first step, the g-membranes also get dissolved by the c-copies in each operational subconfiguration and in the countdown subtree as well.

The following rules are used to dissolve operational subconfigurations:

- $[c]_l \to c$,
- $[e_{1,i}]_{h_{i,j}} \to e_{1,i}$,
- $[e_{i,j}]_{h'_{i,j}} \to e_{1,j}$,
- $[c]_{h_{i,j}} \to c$,
- $[c]_{h'_{i,j}} \to c$,
- $[c]_{l_c} \to \#$

$\Big\}\ i \in [2, n-1], j \in [i+1, n].$

First, notice that the innermost l-membranes during the computation of $\Pi(n)$ always contain a c-copy after the first step. Thus, $\Pi(n)$ dissolves all the innermost l-membranes by applying the rule $[c]_l \to c$ in each computation step. This implies that in each operational subconfiguration, the two embeddings of l-membranes always contain the same number of l-membranes. Consequently, if one embedding of the l-membranes is completely dissolved, then the other embedding of the l-membranes behaves in the same way. That is, a c-copy is sent into both parent membranes and this is done by the same computation step.

Next, we show that no operational subconfiguration op is dissolved until all operational subconfigurations below op are dissolved.

Proposition 2. *Consider the path from the* input*-membrane to the root of the working subtree. For any two different membranes x and y on this path such that $x \rightsquigarrow y$, the following holds.*

(1) If x and y are not in the same operational subconfiguration, then x gets dissolved earlier than y.

(2) If x and y are in the same operational subconfiguration, then y is not dissolved earlier than x.

Proof. First, we prove Statement (1). It clearly holds if x is the input-membrane, so assume that x and y are in different operational subconfigurations. Denote op (resp. op') that operational subconfiguration in which x (resp. y) occurs. Since op has not been dissolved yet, the edge-multiset cannot occur in op'. Consider the step of $\Pi(n)$ when the last l-membranes in op get dissolved. Denote C the configuration of $\Pi(n)$ after this step. It is easy to see that $\Pi(n)$ dissolves all membranes in op in at most three steps. On the other hand, using Equation (3.2), one can see that op' in C must have at least five l-membranes in both embeddings of these membranes. Therefore, it takes for $\Pi(n)$ at least five more steps to dissolve y. This yields that x gets dissolved earlier than y.

Next, we prove Statement (2). Assume that x and y are in the same operational subconfiguration $op_{i,j}$, for some $i \in [2, n-1]$, $j \in [i+1, n]$. Assume on the contrary that y gets dissolved earlier than x. Clearly, the $h'_{i,j}$-membrane must be dissolved earlier than the inner l_c-membrane, and the inner l_c-membrane must be dissolved earlier than the outer l_c-membrane. Therefore, x must be the $h_{i,j}$-membrane and y must be the $h'_{i,j}$-membrane. Since, by our assumption, the $h'_{i,j}$-membrane gets dissolved first, then it must be dissolved by a c-copy. This c-copy must be released from the l-membrane that is initially embedded in the $h'_{i,j}$-membrane. By our note above, the $h_{i,j}$-membrane x must contain a c-copy as well (the c-copy that comes from the l-membranes embedded in the $h_{i,j}$-membrane). Therefore, both x and y get dissolved by the corresponding c-copy in the same computation step contradicting our assumption. □

Looking at the proof of Statement (1) of Proposition 2, one can see the following statement as well.

Corollary 1. *Consider the configuration C, op, and op' defined in the proof of Statement (1) in Proposition 2. Assume that op' is placed right above op in C. Consider that step of Π when the edge-multiset gets into the h-membrane of op'. Then both the h-membrane and the h'-membrane of op' contain at least two l-membranes.*

For every $i \in [2, n-1], j \in [i+1, n]$, denote $\Omega_{i,j}$ (resp. $\hat{\Omega}_{i,j}$) the edge-multiset appearing in the $h_{i,j}$-membrane (resp. appearing in the outer l_c-membrane of $op_{i,j}$). Figure 3.2 shows the operational subconfiguration $op_{i,j}$ right after the appearance of $\Omega_{i,j}$ in the $h_{i,j}$-membrane. By Proposition 2, $\Omega_{i,j}$ and $\hat{\Omega}_{i,j}$ are well defined. Notice that if $op_{i,j}$ is not the outermost operational subconfiguration, then $\hat{\Omega}_{i,j} = \Omega_{i',j'}$, where (i', j') is the index of the operational subconfiguration right above $op_{i,j}$.

Now, we can show that the dissolution of the operational subconfiguration $op_{i,j}$ implements the (i,j)th edge operation of Algorithm 1. To this end, for every $i \in [2, n-1], j \in [i+1, n]$, denote $\mathcal{E}_{i,j}$ (resp. $\hat{\mathcal{E}}_{i,j}$) the set $\hat{\mathcal{E}}_i$ before (resp. after) executing the (i,j)th edge operation of Algorithm 1. Recall that G is a topologically sorted graph with a set of vertices $\mathcal{V}$ and set of edges $\mathcal{E}$. For any $\Delta \subseteq \mathcal{V} \times \mathcal{V}$ and $\Gamma \subseteq \Sigma(n)$,

$$\Delta \approx \Gamma$$

denotes that for every $(l,k) \in \mathcal{V} \times \mathcal{V}$, $(l,k) \in \Delta$ if and only if $e_{l,k} \in \Gamma$.

Proposition 3. *For every $i \in [2, n-1]$ and $j \in [i+1, n]$, $\mathcal{E}_{i,j} \approx \Omega_{i,j}$ implies $\hat{\mathcal{E}}_{i,j} \approx \hat{\Omega}_{i,j}$.*

Proof. Let $i \in [2, n-1]$ and $j \in [i+1, n]$ such that $\mathcal{E}_{i,j} \approx \Omega_{i,j}$. First, assume that either $(1,i) \notin \mathcal{E}_{i,j}$ or $(i,j) \notin \mathcal{E}_{i,j}$ (and thus, $e_{1,i} \notin \Omega_{i,j}$ or $e_{i,j} \notin \Omega_{i,j}$). Then, it is easy to see that $\mathcal{E}_{i,j} = \hat{\mathcal{E}}_{i,j}$. We show that $\Omega_{i,j} = \hat{\Omega}_{i,j}$ which clearly implies $\hat{\mathcal{E}}_{i,j} \approx \hat{\Omega}_{i,j}$. We distinguish the following two cases.

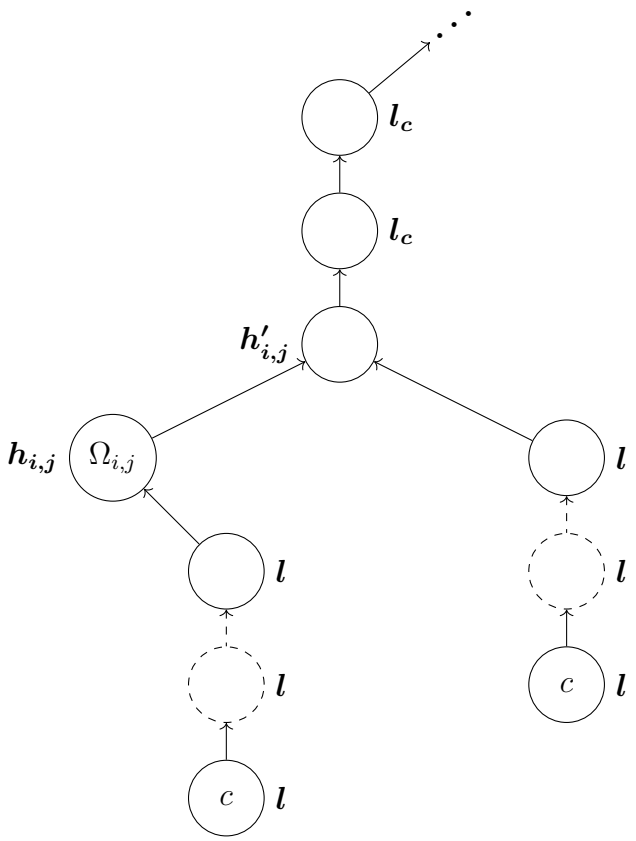


Figure 3.2: *The operational subconfiguration $op_{i,j}$ right after the appearance of the edge-multiset in the $h_{i,j}$-membrane. The nodes and edges drawn with dashed lines indicate that the embeddings of l-membranes contain either 2 or 3 l-membranes.*

- If $e_{1,i} \notin \Omega_{i,j}$, then the $h_{i,j}$-membrane does not get dissolved by the application of the rule $[e_{1,i}]_{h_{i,j}} \to e_{1,i}$. However, the computation continues until a c-copy gets into the $h_{i,j}$-membrane and into the $h'_{i,j}$-membrane from the l-membranes. Then, both of these membranes get dissolved by the corresponding rules and two c-copies along with the edge-multiset $\Omega_{i,j}$ get into the parent membrane of the $h'_{i,j}$-membrane, that is, into the first l_c-membrane. Then, the two l_c-membranes get dissolved step by step by the c-copies which become dummy objects. Consequently, $\hat{\Omega}_{i,j} = \Omega_{i,j}$.

- If $e_{i,j} \notin \Omega_{i,j}$ but $e_{1,i} \in \Omega_{i,j}$, then the $h_{i,j}$-membrane gets dissolved by the application of the rule $[e_{1,i}]_{h_{i,j}} \to e_{1,i}$. That is, the edge-multiset remains the same and gets to the $h'_{i,j}$-membrane. Notice that, by Corollary 1, the $h'_{i,j}$-membrane must exist right after the dissolution of the $h_{i,j}$-membrane. Now, since $e_{i,j} \notin \Omega_{i,j}$, the computation continues similarly as in the previous case. Consequently, $\hat{\Omega}_{i,j} = \Omega_{i,j}$.

Now, assume that $(1,i) \in \mathcal{E}_{i,j}$ and $(i,j) \in \mathcal{E}_{i,j}$ (and thus, $e_{1,i} \in \Omega_{i,j}$ and $e_{i,j} \in \Omega_{i,j}$). In this case, after executing the (i,j)th edge operation, we have that $\hat{\mathcal{E}}_{i,j} = (\mathcal{E}_{i,j} - \{(i,j)\}) \cup \{(1,j)\}$. Using that $e_{1,i} \in \Omega_{i,j}$ and $e_{i,j} \in \Omega_{i,j}$, the computation of $\Pi(n)$ can be described as follows. First the $h_{i,j}$-membrane gets dissolved by the rule $[e_{1,i}]_{h_{i,j}} \to e_{1,i}$. That is, the edge-multiset remains the same and gets to the $h'_{i,j}$-membrane. Notice again that, by Corollary 1, the $h'_{i,j}$-membrane must exist right after the dissolution of the $h_{i,j}$-membrane. Now, the $h'_{i,j}$-membrane still contains an l-membrane, thus it gets dissolved by the rule $[e_{i,j}]_{h'_{i,j}} \to e_{1,j}$. Consequently, $e_{i,j}$ gets replaced by $e_{1,j}$ in the edge-multiset. After this, the edge-multiset remains the same until the dissolution of the last l_c-membrane of $op_{i,j}$. Using that objects of the form $e_{k,l} \in cod(G)$ with $k \neq 1$ can occur at most once in an edge-multiset, we get that $\hat{\Omega}_{i,j}$ is the multiset we get by removing all occurrences of $e_{i,j}$ from $\Omega_{i,j}$ and adding $e_{1,j}$ to the yielded multiset. That is, $\hat{\mathcal{E}}_{i,j} \approx \hat{\Omega}_{i,j}$ which finishes the proof of the statement. □

The following proposition will be useful in the rest of the proof of Theorem 1.

Proposition 4. *The edge-multiset $\hat{\Omega}_{n-1,n}$ appears in the* halt*-membrane in at most $\kappa = \frac{5n^2-15n}{2} + 6$ steps.*

Proof. By Proposition 2, when $op_{n-1,n}$ gets dissolved, all the operational subconfigurations below $op_{n-1,n}$ are already dissolved. One can see that $op_{n-1,n}$ gets dissolved in at most $\kappa = k_{n-1,n} + 4$ steps. By Equation 3.2, $k_{n-1,n} = 2 + (\eta - 1) \cdot 5$, where η is the number of operational configurations in μ. That is, $k_{n-1,n} = 2 + (\frac{n^2-3n+2}{2} - 1) \cdot 5$ and thus $\kappa = \frac{5(n^2-3n+2)}{2} + 1 = \frac{5n^2-15n}{2} + 6$. □

Next, we show that $e_{1,n}$ gets to the halt-membrane if and only if $1 \rightsquigarrow_G n$.

Proposition 5. *During the computation of $\Pi(n)$, the object $e_{1,n}$ appears in the* halt*-membrane if and only if $1 \rightsquigarrow_G n$.*

Proof. By Proposition 1, using that $\mathcal{E}_{n-1} = \hat{\mathcal{E}}_{n-1,n}$, $1 \rightsquigarrow_G n$ if and only if $(1,n) \in \hat{\mathcal{E}}_{n-1,n}$. Since $\mathcal{E}_{2,3} \approx \Omega_{2,3}$, by a repeated application of Proposition 3, we get $e_{1,n} \in \hat{\Omega}_{n-1,n}$ if and only if $(1,n) \in \hat{\mathcal{E}}_{n-1,n}$. Thus, $e_{1,n}$ appears in the halt-membrane if and only if $1 \rightsquigarrow_G n$. □

Now, we describe how the computation goes in the countdown subtree and in the halt-membrane. Here, we use the following rules:

- $[c]_l \to c$,
- $[e_{1,n}]_{\mathsf{halt}} \to yes$,
- $[c]_{\mathsf{halt}} \to no$.

The first rule of the three above ensures that the c-copy in the countdown subtree gets closer to the halt-membrane in each step of the computation. Since the number

of l-membranes in the countdown subtree is $\kappa = \frac{5n^2-15n}{2} + 6$, it takes $\kappa + 1$ steps for $\Pi(n)$ to dissolve the countdown subtree completely (that is, the g-membrane and all the l-membranes). If $\hat{\Omega}_{n-1,n}$ contains $e_{1,n}$, then it triggers the dissolution of the halt-membrane due to the rule $[e_{1,n}]_{\mathsf{halt}} \to yes$, resulting yes in the skin-membrane in at most $\kappa + 1$ steps. Thus, the dissolution of the last l-membrane of the countdown subtree is never done before the dissolution of the halt-membrane triggered by $e_{1,n}$ in this case, avoiding undesired nondeterminism. This results in the appearance of the c-copy coming from the countdown subtree in the skin-membrane. Otherwise, if $e_{1,n} \notin \hat{\Omega}_{n-1,n}$, then this c-copy appears in the halt-membrane after $\kappa + 1$ steps and triggers its dissolution due to the rule $[c]_{\mathsf{halt}} \to no$, resulting in the appearance of no in the skin-membrane. Consequently, we have the following statement.

Proposition 6. *$\Pi(n)$ releases object yes to the* skin*-membrane if objects $e_{1,n}$ appears in the* halt*-membrane. Otherwise, $\Pi(n)$ releases object no to the* skin*-membrane.*

Correctness, L-uniformity, and running time of $\Pi(n)$. The correctness of $\Pi(n)$ follows from Propositions 5 and 6. Using Proposition 4, one can see that the computation of $\Pi(n)$ halts in $O(n^2)$ steps. Moreover, both the encoding of G and the description of $\Pi(n)$ can be computed by a deterministic Turing machine using $O(\log n)$ space. Therefore, $\mathbf{\Pi}$ is an $\mathbf{L}$-uniform family of dissolution P systems capable of solving the problem REACHABILITY problem in polynomial time. With this, we have finished the proof of Theorem 1. □

3.4 Concluding remarks

In this chapter, we investigated the computational power of polarizationless P systems with active membranes employing only dissolution rules. We have seen that these P systems, called dissolution P systems in this chapter, can efficiently solve an $\mathbf{NL}$-complete variant of the well-known REACHABILITY problem using reasonable tight uniformity conditions. However, the exact characterization of the computational power of polynomial-time dissolution P systems is still not defined. If they could solve even $\mathbf{P}$-complete problems, it would imply that dissolution P systems can efficiently solve everything that polarizationless P systems can if Păun's conjecture is true. In the next chapter, we will show that dissolution P systems are able to solve all problems in $\mathbf{P}$ in polynomial time even under very tight uniformity conditions.

Chapter 4

The characterization of P by dissolution P systems

In this chapter, we continue to investigate dissolution P systems as we show that these P systems working in polynomial time characterize $\mathbf{P}$ even under a very tight uniformity condition, namely $\mathbf{DLOGTIME}$-uniformity. The $\mathbf{P}$ upper bound in this result is known, and the $\mathbf{P}$ lower bound will be shown by solving a $\mathbf{P}$-complete variant of the satisfiability problem for Horn formulas.

4.1 Introduction

In the previous chapter, we gave an $\mathbf{NL}$ lower bound to the computational power of polynomial-time dissolution P systems by solving the REACHABILITY problem with them using $\mathbf{L}$-uniformity conditions. In this chapter, we show that dissolution P systems are able to solve efficiently all problems in $\mathbf{P}$ even if much tighter uniformity conditions are used. This will be shown by solving a $\mathbf{P}$-complete variant of the satisfiability problem of Horn formulas, called the HORN3SATNORM problem [15]. This problem was solved in [15] by two restricted variants of P systems with active membranes. One solution used only send-out communication rules, and the other one used only division and dissolution rules. Moreover, in both solutions, the polarizations of the membranes were also used.

To solve HORN3SATNORM, we implement Algorithm 2, which is a variant of the algorithm used in [15] to solve this problem. To implement Lines 6–8 of Algorithm 2, the P systems need to check whether some objects, representing certain clauses and variables of the input formula, occur together in a membrane of the P system or not. This requires a certain amount of context-sensitivity. In Chapter 3, we implemented the necessary context-sensitivity with the use of appropriately timed dissolution of linearly nested membrane structures. The same approach will be used in this chapter to provide context-sensitivity in the constructed P systems.

We consider **DLOGTIME**-uniformity for the presented family of dissolution P systems. **DLOGTIME**-uniformity [21], inspired by Boolean circuit complexity, a uniformity condition presumably weaker than $\mathbf{FAC}^0$-uniformity, which we mentioned in the Introduction of Chapter 3. **DLOGTIME**-uniformity was introduced in [49] for families of P systems. A family $\mathbf{\Pi}$ of P systems is **DLOGTIME**-uniform if both the members of $\mathbf{\Pi}$ and the input encoding can be described by deterministic logarithmic-time Turing machines. Due to their time restrictions, these Turing machines cannot compute the whole representation of a P system but can describe certain local properties of it as we discussed in Chapter 2. Although **DLOGTIME**-uniformity is a rather weak uniformity condition, it is conjectured in [49] that many existing results involving polynomial-time uniform families of P systems can be adapted to **DLOGTIME**-uniformity.

In this chapter, we present a solution of HORN3SATNORM accomplished by a **DLOGTIME**-uniform family of polynomial-time polarizationless P systems with active membranes using no dissolution rules. Since it is known that polarizationless P systems with active membranes using only membrane dissolution rules can solve efficiently only problems in $\mathbf{P}$ [58], this result gives a characterization of the complexity class $\mathbf{P}$ by rather restricted polynomial-time polarizationless P systems under very tight uniformity conditions.

The remainder of the chapter is organized as follows. In the next section, we define **DLOGTIME**-uniformity for dissolution P systems. Then, in Section 4.3, we introduce the HORN3SATNORM problem, and Algorithm 2 that efficiently solves this problem. In Section 4.4, we present an **L**-uniform family $\mathbf{\Pi}$ of dissolution P systems that implements Algorithm 2. In Section 4.5, we present a method that transforms $\mathbf{\Pi}$ into a **DLOGTIME**-uniform family of dissolution P systems that is still able to solve HORN3SATNORM in polynomial time. Finally, in Section 4.6, we make some concluding remarks.

4.2 DLOGTIME-uniformity for dissolution P systems

Consider a dissolution P system of the form (O, H, μ, R). Without loss of generality, we can assume that O contains only objects that appear in the rules of R. Indeed, if O contains an object a that does not appear in any rule of R, then the removal of a from O does not affect the behavior of any computation of these P systems.

Now, we define **DLOGTIME**-uniformity for dissolution P systems by taking Definition 2.3.2 as a basis. Usually, the functions f and cod of Definition 2.3.2 are computed so that the values of these functions appear as outputs of the devices that compute these functions. Since a **DLOGTIME** Turing machine works with an $O(\log n)$ time restriction, the values of f and cod cannot be constructed in this sense during a **DLOGTIME**-computation. For this reason, following Porreca et al. [49], we define

deterministic log-time computable uniformity conditions for membrane systems in a different way. Consider a family of dissolution P systems $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$. Instead of constructing $\Pi(n)$ and the encoding $cod(s)$ of an instance s of a problem D, we will give languages $L^n_{\mathbf{\Pi}}$ and L_D of such words that describe various features of $\Pi(n)$ and $cod(s)$, respectively. These words are of the form $\langle i, \alpha, a_1, \ldots, a_l\rangle$, where $l \geq 1$, $i \in [5]$ is the index of the feature, and for every $j \in [l]$, a_j is a feature-dependent parameter. Moreover, in the case of $L^n_{\mathbf{\Pi}}$, α is 1^n, that is, the index of $\Pi(n)$ in unary notation, while in the case of L_D, α is the input of D. The index of a feature and the feature-dependent parameters, as well as the input of D, are reasonably encoded in these words (see Section 4.5).

Definition 4.2.1. Let $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ be a family of dissolution P systems where $\Pi(n) = (O, H, \mu, R)$ and $\mu = (V, E, L, \omega)$. The *descriptive features* of $\Pi(n)$ and those of the instances of D are defined as follows:

- **Membrane structure.** For every $h_1, h_2 \in H$, $\langle 1, 1^n, h_1, h_2\rangle \in L^n_{\mathbf{\Pi}}$ if and only if for each membrane $y \in V$ with $L(y) = h_1$, there is a membrane $x \in V$ with $L(x) = h_2$ and $(y, x) \in E$. Moreover, for every $h \in H$ and $t \in \mathbb{N}$, $\langle 2, 1^n, h, t\rangle \in L^n_{\mathbf{\Pi}}$ if and only if for each membrane $x \in V$ with $L(x) = h$, x has exactly t child nodes in μ. Notice that in [49], no feature similar to $\langle 2, 1^n, h, t\rangle$ had to be defined since there the nodes of a membrane structure were labeled in a one-to-one way. The defined membrane structure μ must be a tree with a root labeled with skin and the size of μ must be polynomial with respect to n. Notice that due to the polynomial size of μ, the maximal value of t for which $\langle 2, 1^n, h, t\rangle \in L^n_{\mathbf{\Pi}}$ must be bounded by a polynomial with respect to n.
- **Initial multisets.** For every $h \in H$, $i \in \mathbb{N}$, and $a \in O$, $\langle 3, 1^n, h, i, a\rangle \in L^n_{\mathbf{\Pi}}$ if and only if for each membrane $x \in V$ with $L(x) = h$, the ith object of $\omega(x)$ is a. For every $i \geq |\omega(x)|$, $\langle 3, 1^n, h, i, a\rangle$ has to be not included in $L^n_{\mathbf{\Pi}}$, and $|\omega(x)|$ must be bounded by a polynomial with respect to n.
- **Dissolution rules.** For every $h \in H$ and $a, b \in O$, $\langle 4, 1^n, h, a, b\rangle \in L^n_{\mathbf{\Pi}}$ if and only if R contains the rule $[a]_h \to b$.
- **Input multiset.** Let s be an instance of D, $i \in [|cod(s)|]$ and $a \in O$. Consider a fixed order of the objects in $cod(s)$. The word $\langle 5, s, i, a\rangle$ is in L_D if and only if the ith object of $cod(s)$ is a. The size of $cod(s)$ must be polynomial with respect to n.

Notice that in [49], the authors defined two additional features regarding the polynomial upper bounds for the size of O and the size of H. We did not define these features as if the number of objects or labels was greater than $O(n^k)$ then the binary strings encoding objects or labels would not be of $O(\log n)$ length. Thus,

no **DLOGTIME** Turing machine would be able to process the descriptive words defined above.

It can be seen that a membrane structure μ described by the features given in Definition 4.2.1 must fulfill the following properties. Consider two different nodes x, y of μ having the same label. Then (i) x and y must have the same number of child nodes, (ii) the regions of both x and y must contain the same initial multiset, and (iii) the labels of the parent membranes of x and y must be the same. However, this is not a loss of generality as these conditions can be achieved in a dissolution P system by using, for example, one-to-one labeling of the membranes.

Let $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ be a family of dissolution P systems and let D be a decision problem. Moreover, let $L_{\mathbf{\Pi}} = \bigcup_{n=1}^{\infty} L^n_{\mathbf{\Pi}}$. We say that the functions $1^n \mapsto \Pi(n)$ and cod appearing in Definition 2.3.2 are **DLOGTIME**-*computable* if the language $L_{\mathbf{\Pi}} \cup L_D$ is recognizable by a **DLOGTIME** Turing machine. Then the **DLOGTIME**-*uniformity* of $\mathbf{\Pi}$, as well as the decision of D by $\mathbf{\Pi}$ can be defined in the sense of Definition 2.3.2.

4.3 The HORN3SATNORM problem

For the necessary notions of propositional logic that will be used here, we refer to Chapter 2. We say that a formula in CNF is a *Horn-formula* if each of its clauses contains at most one positive literal. The HORNSAT problem sounds as follows: given a Horn-formula φ, decide whether it is satisfiable or not. It is well known that HORNSAT is **P**-complete (see, e.g. [38]). In this chapter, we will consider the HORN3SATNORM problem (H3SN in short) introduced in [15]. In this problem, the input formula is such a Horn formula, where each clause is either a positive unit clause or it contains exactly two negative literals and at most one positive literal. The task is again to determine whether the input formula is satisfiable or not. It was shown in [15] that HORN3SATNORM is **P**-complete as well.

In the rest of the chapter, by a formula we mean an instance of H3SN over Var_n with $n \geq 2$ (for $n < 2$ the task is trivial). Let φ be a formula. Then, by definition, each clause of φ is either a positive unit clause or it is either of the forms $\neg x \vee \neg y$ or $\neg x \vee \neg y \vee z$, where $x, y, z \in Var_n$. By basic equivalences of propositional logic, a clause of the form $\neg x \vee \neg y \vee z$ (resp. $\neg x \vee \neg y$) is equivalent to $x \wedge y \to z$ (resp. $x \wedge y \to \downarrow$), where $\downarrow$ denotes the constant $false$ value. For convenience, by a non-unit clause of φ, we will mean a formula of the form $x \wedge y \to z$ or $x \wedge y \to \downarrow$. Moreover, for the sake of brevity, we will denote $x \wedge y$ by xy in the above formulas.

Let φ be a formula, and let $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$ be an enumeration of all the non-unit clauses over Var_n. We denote the set $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$ by C_n. One can see that $m = |C_n| = O(n^3)$. Moreover, let $Unit^+_\varphi$ be the smallest set in $(Var_n \cup \{\downarrow\})$ satisfying the following properties: (i) all positive unit clauses of φ are in $Unit^+_\varphi$ and (ii) if

$x, y \in Unit^+_\varphi$ and $xy \to z \in \varphi$, then $z \in Unit^+_\varphi$. The set $Unit^+_\varphi$ is useful to decide whether φ is satisfiable according to the following statement, which can easily be proved using basic properties of propositional logic.

Proposition 7. *A formula φ is unsatisfiable if and only if $\downarrow \in Unit^+_\varphi$.*

We will show that $Unit^+_\varphi$ can be calculated by a **DLOGTIME**-uniform family of dissolution P systems. In fact, we will give P systems that implement Algorithm 2 which calculates $Unit^+_\varphi$. To see this, denote $Unit^0_\varphi$ the set of positive unit clauses occurring in φ. Moreover, for every $i \geq 1$, let

$$Unit^i_\varphi = Unit^{i-1}_\varphi \cup \{z \mid \exists xy \to z \in \varphi : x, y \in Unit^{i-1}_\varphi\}.$$

Clearly, $Unit^i_\varphi \subseteq Unit^+_\varphi$ for every $i \geq 0$. Moreover, it is not difficult to see that $Unit^+_\varphi \subseteq Unit^{n-1}_\varphi$. That is,

$$Unit^+_\varphi = Unit^{n-1}_\varphi. \tag{4.1}$$

Using this, we can show the following statement.

Algorithm 2 Decision of H3SN

```
1:  function H3SN(φ)                          ▷ φ is an instance of H3SN with n variables
2:      Δ_0 = Unit^0_φ
3:      for i = 1 ... n − 1 do
4:          Θ_i = Δ_{i−1}
5:          for j = 1 ... m do                                          ▷ m = |C_n|
6:              if C_j = xy → z ∈ φ and x, y ∈ Θ_i then   ▷ x, y ∈ Var_n, z ∈ Var_n ∪ {↓}
7:                  Add(Θ_i,z)                                  ▷ extending Θ_i by z
8:              end if
9:          end for
10:         Δ_i = Θ_i
11:     end for
12:     if ↓ ∈ Δ_{n−1} then
13:         return false
14:     else
15:         return true
16:     end if
17: end function
```

Proposition 8. *$Unit^+_\varphi = \Delta_{n-1}$ where Δ_{n-1} is the set appearing in Line 12 of Algorithm 2.*

Proof. By Equation (4.1), we know that $Unit^+_\varphi = Unit^{n-1}_\varphi$. Moreover, clearly, $\Delta_{n-1} \subseteq Unit^+_\varphi$. Thus, to see the statement, it is enough to show that for every $i \in [0, n-1]$,

$$Unit^i_\varphi \subseteq \Delta_i, \tag{4.2}$$

where Δ_i is defined as follows. If $i = 0$, then Δ_i is the set defined in Line 2 of Algorithm 2. Otherwise, Δ_i is the set appearing in Line 10 of Algorithm 2 (notice that if $i = n - 1$, then the sets Δ_{n-1} appearing in Line 10 and in Line 12 are the same).

We show Expression (4.2) by induction on i. If $i = 0$, then Expression (4.2) holds since $\Delta_0 = Unit^0_\varphi$. Assume that Expression (4.2) holds when $i \in [n-2]$, we show that it also holds for $i+1$. Assume on the contrary that $Unit^{i+1}_\varphi \not\subseteq \Delta_{i+1}$ and let $z \in Unit^{i+1}_\varphi - \Delta_{i+1}$. By the induction hypothesis, $Unit^i_\varphi \subseteq \Delta_i$. Moreover, clearly, $\Delta_i \subseteq \Delta_{i+1}$. Thus, $z \notin Unit^i_\varphi$. Then, by the definition of $Unit^{i+1}_\varphi$, there are $x, y \in Unit^i_\varphi$ such that $xy \to z \in \varphi$. Since $Unit^i_\varphi \subseteq \Delta_i$, $x, y \in \Delta_i$. Then $x, y \in \Theta_{i+1}$, where Θ_{i+1} is the set defined in Line 4 of Algorithm 2. Therefore, by Lines 6-8, z is added to Θ_{i+1}. Consequently, $z \in \Delta_{i+1}$ by Line 10. This is a contradiction, which proves Expression (4.2). □

In the rest of the chapter, we will call the execution of Lines 6 and 7 in Algorithm 2 for a clause $xy \to z \in C_n$ in the ith iteration of the outer cycle the *ith conditional extension by $xy \to z$*.

4.4 Solving HORN3SATNORM with dissolution P systems

In this section, we prove the following main result of the chapter.

Theorem 2. *The* HORN3SATNORM *problem can be solved in polynomial time by a* **DLOGTIME***-uniform family of dissolution P systems.*

The rest of this chapter is devoted to the proof of this theorem, which is divided into three parts. First, we give an **L**-uniform family $\mathbf{\Pi} = \{\Pi(n) \mid n \geq 2\}$ of dissolution P systems in Subsection 4.4.1, where $\Pi(n)$ is devoted to decide whether an instance of H3SN over Var_n is satisfiable or not by simulating Algorithm 2. Then, in Subsection 4.4.2, we give an overview of the computation of $\Pi(n)$ and show its correctness. Finally, in Section 4.5, we show that $\mathbf{\Pi}$ can be modified so that it can be described in a **DLOGTIME**-uniform way.

4.4.1 The definition of Π

Let us fix for the rest of this section a formula φ over Var_n. First, we define an alphabet to encode φ:

$$\Sigma(n) = \{v_x, \overline{v}_x \mid x \in Var_n \cup \{\downarrow\}\} \cup \{c_{xy\to z} \mid xy \to z \in C_n\}$$

An object $\overline{v}_x \in \Sigma(n)$ is called the *complement* of v_x. Let $\Gamma \subseteq \Sigma(n)$. We call Γ *complete*, if for every $x \in Var_n \cup \{\downarrow\}$, Γ contains either v_x or its complement, but not both.

Moreover, for a set $\Theta \subseteq Var_n \cup \{\downarrow\}$ and a complete set $\Gamma \subseteq \Sigma(n)$, denote $\Theta \approx \Gamma$ that for every $x \in Var_n \cup \{\downarrow\}$,

$$x \in \Theta \iff v_x \in \Gamma.$$

The encoding of φ, denoted by $cod(\varphi)$, is the following complete subset of $\Sigma(n)$:

$$cod(\varphi) = \{v_x \mid x \in \varphi\} \cup \{\overline{v}_x \mid x \notin \varphi\} \cup \{c_{xy\to z} \mid xy \to z \in \varphi\} \cup \{\overline{v}_\downarrow\}.$$

Now, the P system $\Pi(n) = (O, H, \mu, R)$ is defined as follows:

- $O = \Sigma(n) \cup \{d, \hat{d}, yes, no\}$,
- $H = \{l_{xy\to z}, l'_{xy\to z}, l''_{xy\to z}, l'''_{xy\to z} \mid xy \to z \in C_n\} \cup \{\mathsf{skin}, \mathsf{check}, \mathsf{input}, g, l, l_d\}$,
- the set $R = R_1 \cup R_2 \cup R_3$ of rules of $\Pi(n)$ is given in Table 4.1,
- the input membrane and the output membrane are labeled with input and skin, respectively, and
- the initial membrane configuration μ is defined below.

- $[d]_{\mathsf{input}} \to \hat{d}$
- $[d]_g \to d$

(a) *The set of rules* R_1

- $[\overline{v}_\downarrow]_{\mathsf{check}} \to yes$
- $[v_\downarrow]_{\mathsf{check}} \to no$

(b) *The set of rules* R_3

- $[d]_l \to d$
- $[c_{xy\to z}]_{l_{xy\to z}} \to c_{xy\to z}$
- $[v_x]_{l'_{xy\to z}} \to v_x$
- $[v_y]_{l''_{xy\to z}} \to v_y$
- $[\overline{v}_z]_{l'''_{xy\to z}} \to v_z$
- $[v_z]_{l'''_{xy\to z}} \to v_z$

- $[d]_{l_{xy\to z}} \to d$
- $[d]_{l'_{xy\to z}} \to d$
- $[d]_{l''_{xy\to z}} \to d$
- $[d]_{l'''_{xy\to z}} \to d$
- $[d]_{l_d} \to \hat{d}$

(c) *The set of rules* R_2 *(*$x, y \in Var_n, z \in Var_n \cup \{\downarrow\}$*)*

Table 4.1: *The three subsets of* R.

We define $\mu = (V, E, L, \omega)$ with the help of Figures 4.1a, 4.1b and 4.2 as follows. The rectangles and circles denote subconfigurations and membranes of μ, respectively. An edge leading from a rectangle is connected to the root of the subconfiguration represented by that rectangle. The name of a subconfiguration is placed beside the corresponding rectangle. We depict the labels of the membranes similarly. Moreover, the multiset $\omega(x)$ associated initially with a membrane x is written in the corresponding circle. Empty circles represent empty multisets. The root of the membrane structure is the skin-membrane, which has an only child with label check. Initially, these membranes are empty.

Recall that $m = |C_n|$. The check-membrane contains $(n-1) \cdot m$ subconfigurations that are nested in each other (see Figure 4.1). These subconfigurations are denoted

by $ext^i_{\mathcal{C}_j}$, $i \in [n-1], j \in [m]$, respectively, and $ext^i_{\mathcal{C}_j}$ corresponds to the ith conditional extension by $\mathcal{C}_j$ of Algorithm 2. Thus, we will call these subconfigurations *extensional subconfigurations*. Moreover, we call the subconfiguration consisting of $ext^i_{\mathcal{C}_1}, \ldots, ext^i_{\mathcal{C}_m}$ the *ith block*.

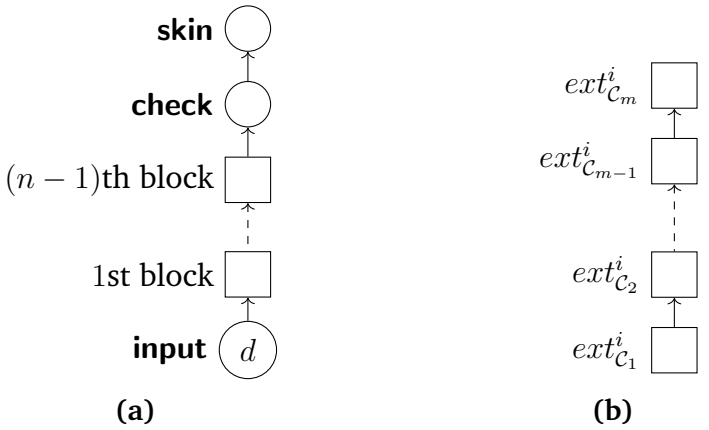


Figure 4.1: *The initial membrane configuration (a) and the ith block with $i \in [n-1]$ (b). The rectangles denote subconfigurations, the circles denote membranes.*

Let $xy \to z = \mathcal{C}_j$ for some $j \in [m]$. Figure 4.2 shows the extensional subconfiguration $ext^i_{xy\to z}$ of μ for some $i \in [n-1]$. The four outermost membranes in $ext^i_{xy\to z}$ are l_d-membranes. The innermost l_d-membrane contains four nested membranes labeled with $l'''_{xy\to z}$, $l''_{xy\to z}$, $l'_{xy\to z}$, and $l_{xy\to z}$, respectively, which we will call *clause-membranes* in what follows. If $i \neq 1$ or $j \neq 1$, then the $l_{xy\to z}$-membrane contains the next extensional subconfiguration below $ext^i_{xy\to z}$ according to Figure 4.1. If $i = 1$ and $j = 1$, then the $l_{xy\to z}$-membrane contains the input-membrane of $\Pi(n)$. Moreover, $\omega(s) = d$, that is, the input-membrane initially contains a d-copy. Furthermore, for every $i \in [n-1], j \in [m]$, each clause-membrane of $ext^i_{\mathcal{C}_j}$ contains a membrane structure of linearly nested l-membranes. We call this structure an *l-structure* and by the *depth* of an l-structure we mean the number of l-membranes in this l-structure. The depth $k_{i,j}$ of each l-structure is defined recursively as follows:

$$k_{i,j} = \begin{cases} 4, \text{ if } i = 1, j = 1, \\ 9 + k_{i,j-1}, \text{ if } i \in [n-1], j \in [2,m], \\ 9 + k_{i-1,m}, \text{ if } i \in [2,n-1], j = 1. \end{cases} \tag{4.3}$$

Finally, each of the innermost l-membranes in $ext^i_{\mathcal{C}_j}$ contains an elementary g-membrane initially containing a d-copy. For any membrane s in $ext^i_{\mathcal{C}_j}$ with $L(s) \neq g$, $\omega(s) = \varepsilon$.

Consider an extensional subconfiguration ext. By saying that $\Pi(n)$ *dissolves* ext, we mean that $\Pi(n)$ dissolves all the membranes initially placed in ext.

Figure 4.3 shows the initial membrane configuration of $\Pi(3)$, where the exten-

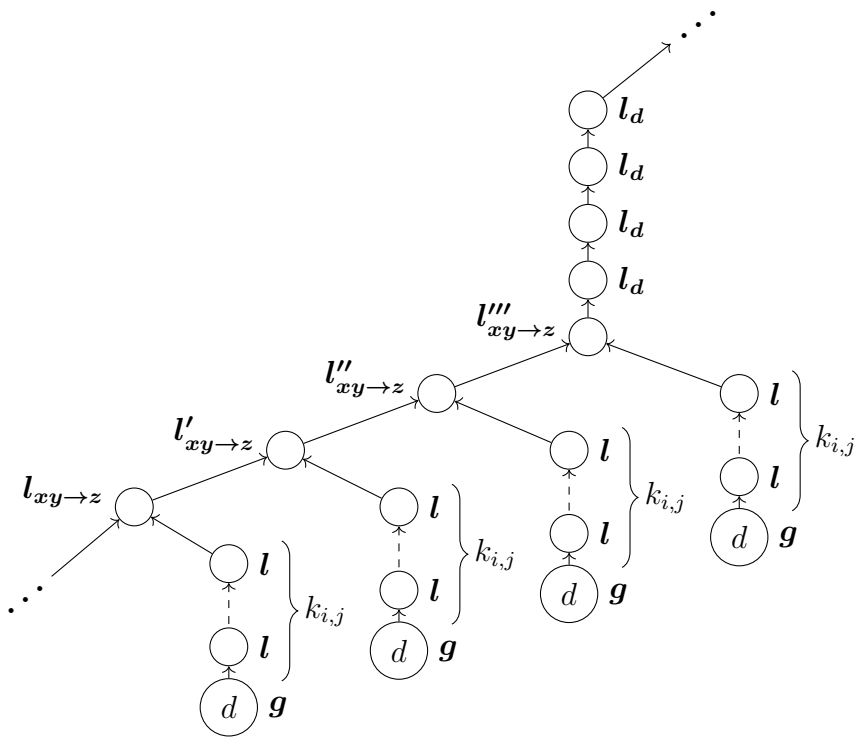


Figure 4.2: *The subconfiguration $ext^i_{xy\to z}$ of μ where $xy \to z = C_j$ for some $j \in [m]$.*

sional subconfigurations are nested according to the following order of all non-unit clauses over variables in Var_3:

$$x_1x_2 \to x_3,\ x_1x_2 \to\downarrow,\ x_1x_3 \to x_2,\ x_1x_3 \to\downarrow,\ x_2x_3 \to x_1,\ x_2x_3 \to\downarrow .$$

Since $n = 3$ and thus, $m = 6$, the membrane structure contains two blocks and both blocks contain six extensional subconfigurations.

4.4.2 An overview of the computation

Assume that $\Pi(n)$ is started with the multiset $cod(\varphi)$ in the input-membrane. We describe the computation of $\Pi(n)$ and along with it, we also show the correctness of $\Pi(n)$.

First, notice that dissolution rules do not change the total number of objects in a membrane structure. Therefore, the number of objects derived from the objects in $cod(\varphi)$ is $|cod(\varphi)|$ during the whole computation. Moreover, these objects occur in the same region in each configuration of the computation. In what follows, we call a multiset that contains the objects derived from $cod(\varphi)$ a *clause-multiset*. For brevity,

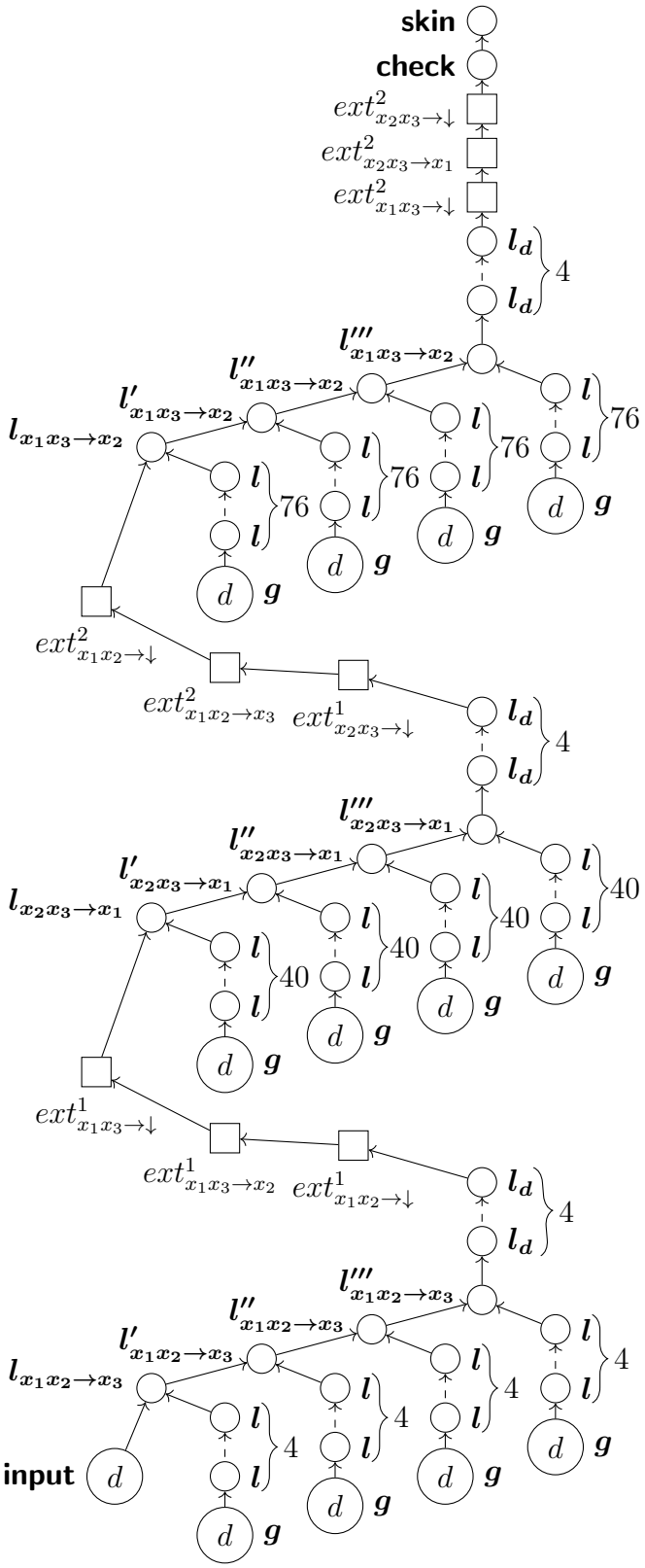


Figure 4.3: *The initial membrane structure of* $\Pi(3)$. *The rectangles denote subconfigurations, the circles denote membranes. The extensional subconfigurations* $ext^1_{x_1x_2\to x_3}, ext^1_{x_2x_3\to x_1}$ *and* $ext^2_{x_1x_3\to x_2}$ *are illustrated in more detail.*

if an object of a clause-multiset M dissolves a membrane s, we will often say that M *dissolves* s. Inspecting the rules in R and using that $cod(\varphi)$ is complete, one can see that the clause-multisets are complete and contain $c_{xy\to z}$ if and only if $c_{xy\to z} \in cod(\varphi)$.

To see how $\Pi(n)$ implements Algorithm 2, we first describe the intuition behind the construction of μ. As already mentioned, an extensional subconfiguration $ext^i_{xy\to z}$ is designed to implement the ith conditional extension by the clause $xy \to z$ of Algorithm 2. Therefore, we constructed $ext^i_{xy\to z}$ to handle the following two main cases: (1) when the clause-multiset appears in the $l_{xy\to z}$-membrane of $ext^i_{xy\to z}$, then it contains all the objects $c_{xy\to z}, v_x, v_y$ and $\overline{v}_z$, and (2) at least one of the objects $c_{xy\to z}, v_x, v_y$ is missing from the clause-multiset appearing in the $l_{xy\to z}$-membrane. In both cases, the l-structures in $ext^i_{xy\to z}$ are dissolved completely, and four d-copies will appear either in the clause-membranes or in the l_d-membranes. Both the clause-membranes and the l_d-membranes can be dissolved by these copies. However, the l_d-membranes can be dissolved only by a d-copy.

In case (1), the objects $c_{xy\to z}, v_x, v_y$ and $\overline{v}_z$ dissolve all clause-membranes of $ext^i_{\mathcal{C}_j}$, respectively, and, meanwhile, object $\overline{v}_z$ is replaced by an object v_z. The size of the l-structures is calculated so that the d-copies produced by the l-structures appear in the innermost l_d-membranes in this case.

On the other hand, in case (2), one of the clause-membranes cannot be dissolved by the clause-multiset. Therefore, v_z is not introduced, and those clause-membranes that are not dissolved by the clause-multiset will be dissolved by the d-copies produced by the l-structures.

Next, we describe in detail the computation of $\Pi(n)$. In the first step, $\Pi(n)$ uses only the rules of R_1 (see Table 4.1a). Since the input-membrane initially contains a d-copy, it gets dissolved by the application of the first rule in R_1. After applying this rule, d becomes $\hat{d}$, a dummy object, that will not be used anymore. Therefore, when we talk about the content of a region, we will not consider these dummy objects. Thus, we can say that after the first step, the $l_{xy\to z}$-membrane of $ext^1_{\mathcal{C}_1}$ contains only $cod(\varphi)$. Still in the first step, the g-membranes also get dissolved by the d-copies they contain in each extensional subconfiguration.

The rules of R_2 (see Table 4.1c) are used to dissolve extensional subconfigurations. To describe how the dissolution of these subconfigurations implements conditional extensions of Algorithm 2 we need some preparation.

First, we make the following observations. After the first step of $\Pi(n)$, the elementary l-membranes contain a d-copy during the whole computation. Thus, $\Pi(n)$ dissolves all the innermost l-membranes by applying the rule $[d]_l \to d$ in each computation step. This implies that in each extensional subconfiguration ext, the four l-structures always have the same depth. Consequently, all l-structures of ext get dissolved in the same computation step and send a d-copy into their parent membranes, respectively. Moreover, these d-copies can dissolve the clause-membranes and also

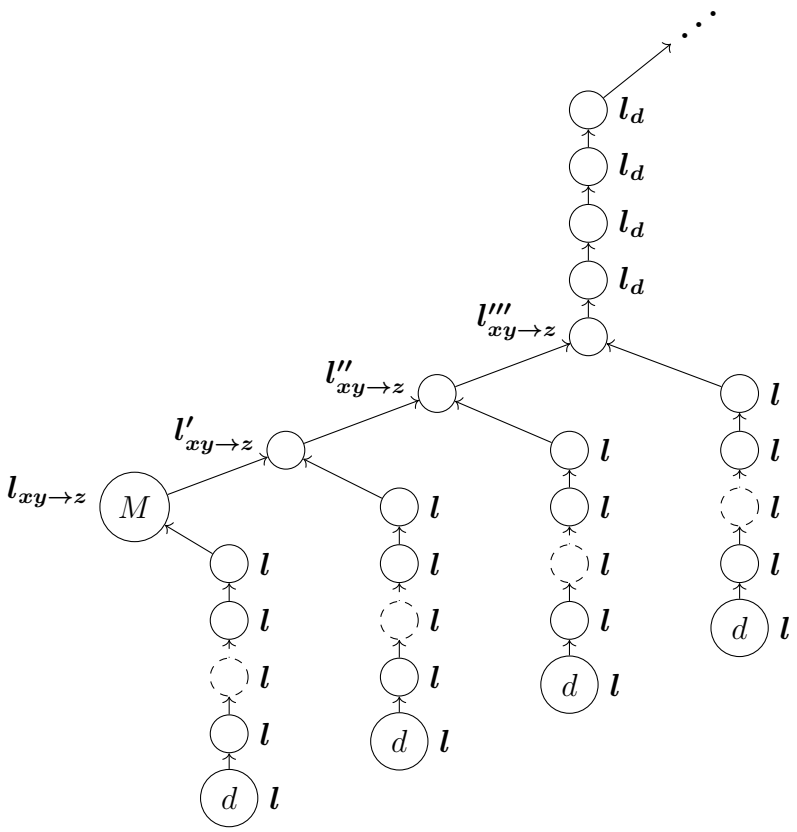


Figure 4.4: *The extensional subconfiguration $ext^i_{xy\to z}$ right after the appearance of the clause-multiset M in the $l_{xy\to z}$-membrane. The nodes and edges drawn with dashed lines indicate that the depth of the l-structures is either 4 or 5.*

the l_d-membranes of ext using the rules in the second column in Table 4.1c. In fact, the l_d-membranes of ext can be dissolved only by a d-copy. Furthermore, if a d-copy dissolves an l_d-membrane, then it becomes a dummy object (cf. the rule $[d]_{l_d} \to \hat{d}$ in R_2), and therefore it cannot be used to dissolve the membranes of other extensional subconfigurations. Using these observations and taking into consideration the depth of the l-structures defined by Expression (4.3), one can see the following.

Proposition 9. *Let $i \in [n-1], j \in [m], \mathcal{C}_j = xy \to z$ and consider the extensional subconfiguration $ext^i_{xy\to z}$. The following holds.*

(1) During the computation, a clause-multiset appears in the $l_{xy\to z}$-membrane and in the outermost l_d-membrane of $ext^i_{xy\to z}$.

(2) When a clause-multiset appears in the $l_{xy\to z}$-membrane of $ext^i_{xy\to z}$, then the depth of the l-structures of $ext^i_{xy\to z}$ is at least four (see also Figure 4.4).

(3) When a clause-multiset appears in the outermost l_d-membrane of $ext^i_{xy\to z}$, then exactly one d-copy appears along with it, and this l_d-membrane is a leaf of the

current membrane structure.

Proof. First, consider the totally ordered set $(EXT, \preceq)$ of extensional subconfigurations, where $EXT = \{ext^i_{\mathcal{C}_j} \mid i \in [n-1], j \in [m]\}$, and for all $ext^{i_1}_{\mathcal{C}_{j_1}}, ext^{i_2}_{\mathcal{C}_{j_2}} \in EXT$, $ext^{i_1}_{\mathcal{C}_{j_1}} \preceq ext^{i_2}_{\mathcal{C}_{j_2}}$, if $ext^{i_1}_{\mathcal{C}_{j_1}}$ is not above $ext^{i_2}_{\mathcal{C}_{j_2}}$ in μ.

We prove Statements (1), (2), and (3) by induction on $ext^i_{\mathcal{C}_j} \in (EXT, \preceq)$ with $\mathcal{C}_j = xy \to z$. First, we show that Statements (1), (2) and (3) hold if $ext^i_{\mathcal{C}_j} = ext^1_{\mathcal{C}_1}$. In the first step of the computation, the input-membrane and the g-membranes get dissolved by the application of the rules of R_1. This results in the appearance of the clause-multiset in the $l_{xy\to z}$-membrane of $ext^1_{\mathcal{C}_1}$, which yields that the first part of Statement (1) holds for $ext^1_{\mathcal{C}_1}$. Moreover, in the first step, the depth of the l-structures does not change. Thus, by Equation (4.3), the depth of the l-structures in $ext^1_{\mathcal{C}_1}$ is exactly four. Hence, Statement (2) holds as well for $ext^1_{\mathcal{C}_1}$.

Looking at the rules of R_2, one can see that $ext^1_{\mathcal{C}_1}$ can be dissolved in two ways:

(a) each clause-membrane gets dissolved by the clause-multiset or

(b) the clause-multiset cannot dissolve every clause-membrane.

In both cases, due to the rule $[d]_l \to d$, the l-structures completely get dissolved after four steps. If all of the clause-membranes are already dissolved by the clause-multiset, then their dissolution took four steps, and the four d-copies appear in the innermost l_d-membrane immediately after the dissolution of the l-structures. Clearly, at this point, this l_d-membrane contains also the clause-multiset. Moreover, the innermost l_d-membrane becomes an elementary membrane when the four d-copies reach it. After three steps, the three innermost l_d-membranes get dissolved by the employment of the only rule that is applicable to the l_d-membranes, $[d]_{l_d} \to \hat{d}$. As a result of this, the outermost l_d-membrane becomes an elementary membrane, and it contains the clause-multiset along with the only remaining d-copy. Thus, in case (a) Statement (3) and the second part of Statement (1) hold.

We note here that the application of the rule $[d]_{l_d} \to \hat{d}$ makes each d-copy in $ext^1_{\mathcal{C}_1}$ evolve to $\hat{d}$. Therefore, no d-copy, that is initially placed in $ext^1_{\mathcal{C}_1}$, has any effect to any extensional subconfiguration above $ext^1_{\mathcal{C}_1}$.

Next, assume that not all of the clause-membranes get dissolved by the clause-multiset by the time the l-structures get completely dissolved (that is, case (b) holds). Then all d-copies appear in the remaining clause-membranes. In the next step, they trigger the corresponding rules occurring in the second column of Table 4.1c. This results in the appearance of the four d-copies and the clause-multiset in the innermost l_d-membrane. From here the dissolution of the l_d-membranes goes exactly the same way as in case (a). Hence, Statement (3) and the second part of Statement (1) hold in case (b) as well.

Now, consider $ext^i_{\mathcal{C}_j}$ with $i \in [n-1], j \in [m]$ and $ext^i_{\mathcal{C}_j} \neq ext^1_{\mathcal{C}_1}$. Let ext be the extensional subconfiguration right below $ext^i_{\mathcal{C}_j}$ and assume that Statements (1), (2)

and (3) hold for ext. We show that Statements (1), (2) and (3) hold for $ext^i_{\mathcal{C}_j}$ as well. Consider the membrane configuration we get right after the dissolution of the l-structures of ext. Due to Equation (4.3) and the continuous application of the rule $[d]_l \to d$ from step 2, the depth of the l-structures in $ext^i_{\mathcal{C}_j}$ is nine in this configuration. From here, it takes at most five steps for $\Pi(n)$ to completely dissolve ext. Thus, the depth of the l-structures in $ext^i_{\mathcal{C}_j}$ is at least four right after the dissolution of ext. This implies that the $l_{xy\to z}$-membrane of $ext^i_{\mathcal{C}_j}$ does not get dissolved as far as ext is not dissolved. Moreover, by Statement (3), there is a membrane configuration in which the outermost l_d-membrane of ext is an elementary membrane and contains the clause-multiset along with a d-copy. Employing the only applicable rule for this l_d-membrane, $[d]_{l_d} \to \hat{d}$, results the appearance of the clause-multiset in the $l_{xy\to z}$-membrane of $ext^i_{\mathcal{C}_j}$. Hence, the first part of Statement (1) and Statement (2) hold for $ext^i_{\mathcal{C}_j}$ as well.

Let μ' denote the membrane configuration we get right after the dissolution of the last l_d-membrane of ext. Using the rules of R_2 and the similarity between $ext^1_{\mathcal{C}_1}$ after the first step and $ext^i_{\mathcal{C}_j}$ in μ', one can see that the computation goes similarly in $ext^i_{\mathcal{C}_j}$ from μ' as in $ext^1_{\mathcal{C}_1}$ after the first step. Thus, the cases (a) and (b) discussed above hold in $ext^i_{\mathcal{C}_j}$ as well. Hence, Statement (3) and the second part of Statement (1) hold for $ext^i_{\mathcal{C}_j}$. □

Let $i \in [n-1], j \in [m]$ and $\mathcal{C}_j = xy \to z \in C_n$. Denote $\Delta_{i,j}$ (resp. $\hat{\Delta}_{i,j}$) the set Θ_i before (resp. after) executing the ith conditional extension by $\mathcal{C}_j$ of Algorithm 2. Likewise, denote $\Omega_{i,j}$ (resp. $\hat{\Omega}_{i,j}$) the clause-multiset appearing in the $l_{xy\to z}$-membrane (resp. appearing in the outermost l_d-membrane) of $ext^i_{\mathcal{C}_j}$. By Statement (1) of Proposition 9, $\Omega_{i,j}$ and $\hat{\Omega}_{i,j}$ are well defined. Moreover, if $i \neq n-1$ and $j \neq m$, then $\hat{\Omega}_{i,j} = \Omega_{i',j'}$, where (i', j') is the index of the extensional subconfiguration right above $ext^i_{\mathcal{C}_j}$. Notice that for the multiset M appearing in Figure 4.4, $M = \Omega_{i,j}$.

Next, we show that the dissolution of $ext^i_{\mathcal{C}_j}$ implements the ith conditional extension by $\mathcal{C}_j$ of Algorithm 2.

Proposition 10. *For each $i \in [n-1]$ and $j \in [m]$,*

$$\Delta_{i,j} \approx \Omega_{i,j} \textit{ implies } \hat{\Delta}_{i,j} \approx \hat{\Omega}_{i,j}.$$

Proof. Let $i \in [n-1]$, $j \in [m]$ and $\mathcal{C}_j = xy \to z \in C_n$ such that $\Delta_{i,j} \approx \Omega_{i,j}$. Then, for every $x \in Var_n \cup \{\downarrow\}$, $x \in \Delta_{i,j}$ if and only if $v_x \in \Omega_{i,j}$. Furthermore, since $\Omega_{i,j}$ is complete, $v_x \in \Omega_{i,j}$ if and only if $\overline{v}_x \notin \Omega_{i,j}$. Recall that when $\Omega_{i,j}$ appears in the $l_{xy\to z}$-membrane of $ext^i_{xy\to z}$, then the depth of l-structures in $ext^i_{xy\to z}$ is at least four by Statement (2) of Proposition 9.

Assume first that $\Delta_{i,j} = \hat{\Delta}_{i,j}$. Then, in Line 6 of Algorithm 2, either z is not added

to $\Delta_{i,j}$ or $z \in \Delta_{i,j}$. That is, we have the following cases:

$$1.\ xy \to z \notin \varphi,\ 2.\ x \notin \Delta_{i,j},\ 3.\ y \notin \Delta_{i,j},\ 4.\ z \in \Delta_{i,j}.$$

We show that in each case $\Omega_{i,j} = \hat{\Omega}_{i,j}$ implying that $\hat{\Delta}_{i,j} \approx \hat{\Omega}_{i,j}$. We can assume that when we check a case $k \in [4]$, none of the cases before k holds.

Case 1: In this case, $c_{xy\to z} \notin cod(\varphi)$, and thus the rule $[c_{xy\to z}]_{l_{xy\to z}} \to c_{xy\to z}$ cannot be applied. Consequently, the four clause-membranes of $ext^i_{xy\to z}$ get dissolved in the very same step by those d-copies that appear in them after the dissolution of the l-structures. This results the appearance of the clause-multiset in the innermost l_d-membrane, and each of the l_d-membranes gets dissolved by the application of the rule $[d]_{l_d} \to \hat{d}$.

Case 2: After that the $l_{xy\to z}$-membrane gets dissolved by the application of the rule $[c_{xy\to z}]_{l_{xy\to z}} \to c_{xy\to z}$, the clause-multiset appears in the $l'_{xy\to z}$-membrane. Since $v_x \notin \Omega_{i,j}$, the rule $[v_x]_{l'_{xy\to z}} \to v_x$ cannot be applied, so the three remaining clause-membranes and the l_d-membranes get dissolved by the d-copies.

Case 3: This case is similar to the previous one. However now, by employing the rules $[c_{xy\to z}]_{l_{xy\to z}} \to c_{xy\to z}$ and $[v_x]_{l'_{xy\to z}} \to v_x$, the corresponding membranes got dissolved in succession, and the clause-multiset appears in the $l''_{xy\to z}$-membrane. Here, since $v_y \notin \Omega_{i,j}$, the rule $[v_y]_{l''_{xy\to z}} \to v_y$ cannot be employed. Hence, the remaining two clause-membranes of $ext^i_{xy\to z}$, as well as the l_d-membranes, get dissolved by the d-copies.

Case 4: In this case each clause-membrane gets dissolved by applying the rules $[c_{xy\to z}]_{l_{xy\to z}} \to c_{xy\to z}, [v_x]_{l'_{xy\to z}} \to v_x, [v_y]_{l''_{xy\to z}} \to v_y$ and $[v_z]_{l'''_{xy\to z}} \to v_z$, one after another. After this, the l_d-membranes get dissolved by the d-copies.

Since the objects of the clause-multiset remain the same in all of the above cases, our statement follows.

Next, assume that $\Delta_{i,j} \neq \hat{\Delta}_{i,j}$. Then, the following holds:

$$xy \to z \in \varphi, x \in \Delta_{i,j}, y \in \Delta_{i,j}, z \notin \Delta_{i,j} \text{ and } z \in \hat{\Delta}_{i,j}.$$

That is, all the conditions of Line 6 in Algorithm 2 are met, but $z \notin \Theta_i$, thus Θ_i gets extended by z. Moreover, $v_x, v_y \in \Omega_{i,j}$ and $v_z \notin \Omega_{i,j}$ by assumption. Furthermore, $\overline{v}_z \in \Omega_{i,j}$ as $\Omega_{i,j}$ is complete. Consider the membrane configuration right after the application of the rules $[c_{xy\to z}]_{l_{xy\to z}} \to c_{xy\to z}, [v_x]_{l'_{xy\to z}} \to v_x$ and $[v_y]_{l''_{xy\to z}} \to v_y$. Here, we have one clause-membrane remaining, the $l'''_{xy\to z}$-membrane. Since it took three steps to employ the rules above, the depth of the l-structures is at least one. Thus, the only rule that is applicable for the $l'''_{xy\to z}$-membrane is $[\overline{v}_z]_{l'''_{xy\to z}} \to v_z$, and its application results the substitution of the object $\overline{v}_z$ by the object v_z in the clause-multiset. Consequently, $\hat{\Delta}_{i,j} \approx \hat{\Omega}_{i,j}$. □

Now, we have all the ingredients necessary to show the correctness of $\Pi(n)$. Assume that $\Pi(n)$ is started with $cod(\varphi)$ in its input membrane. By Propositions 7 and 8, φ is unsatisfiable if and only if $\downarrow \in \Delta_{n-1}$. By definition, $\Delta_{n-1} = \hat{\Delta}_{n-1,m}$. Recall that, for each $i \in [n-1], j \in [m]$, if $i \neq n-1$ and $j \neq m$, then $\hat{\Delta}_{i,j} = \Delta_{i',j'}$ and $\hat{\Omega}_{i,j} = \Omega_{i',j'}$, where (i', j') is defined as follows:

$$(i', j') = \begin{cases} (i, j+1), \text{ if } i \in [n-1], j \in [m-1], \\ (i+1, 1), \text{ if } i \in [n-2], j = m. \end{cases}$$

Since $\Delta_{1,1} = \Delta_0 \approx cod(\varphi) = \Omega_{1,1}$, by a repeated application of Proposition 10, $\hat{\Delta}_{n-1,m} \approx \hat{\Omega}_{n-1,m}$. Therefore, φ is unsatisfiable if and only if $v_\downarrow \in \hat{\Omega}_{n-1,m}$. Thus, also using that $\hat{\Omega}_{n-1,m}$ is complete, $v_\downarrow$ appears in the check-membrane if φ is unsatisfiable, and $\overline{v}_\downarrow$ appears in the check-membrane if φ is satisfiable. Then, the correctness of $\Pi(n)$ follows by the rules of R_3.

By Statement (3) of Proposition 9, when $ext^{n-1}_{\mathcal{C}_m}$ gets dissolved, all the extensional subconfigurations below $ext^{n-1}_{\mathcal{C}_m}$ are already dissolved. One can see that $ext^{n-1}_{\mathcal{C}_m}$ gets dissolved in at most $k_{n-1,m} + 6$ steps. By Equation (4.3), $k_{n-1,m} = O(nm)$. Since $m = O(n^3)$, we get that $\Pi(n)$ halts in $O(n^4)$ steps.

4.5 DLOGTIME-uniformity of $\mathbf{\Pi}$

Now, we discuss the **DLOGTIME**-uniformity of $\mathbf{\Pi}$. Recall that $\Pi(n) = (O, H, \mu, R)$, $Var_n = \{x_1, x_2, \ldots, x_n\}$, and $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$ is an enumeration of all the non-unit clauses over Var_n. Recall also that C_n denotes the set $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$ and $L^n_{\mathbf{\Pi}}$ denotes the language of descriptive words regarding $\Pi(n)$ (see Definition 4.2.1). First, we need to change the labeling of the membranes of μ due to the following reason. Consider the labeling of the membranes appearing in Figure 4.2. It is not difficult to see that using these labels, the word $\langle 1, 1^n, l_d, l_d \rangle$ must be in $L^n_{\mathbf{\Pi}}$. But then $L^n_{\mathbf{\Pi}}$ also defines such a membrane structure μ' where the number of l_d-membranes in an extensional subconfiguration is not exactly four, which contradicts our aim, that $L^n_{\mathbf{\Pi}}$ defines $\mathbf{\Pi}$ unequivocally. Therefore, to be able to show the **DLOGTIME**-uniformity of $\mathbf{\Pi}$, we need to use more informative labels for the membranes in μ than those we have used so far.

To define the appropriate labeling of the membranes in μ, we first have to define an order of the clauses in C_n. Notice that neither in Algorithm 2 nor in the definition of $\Pi(n)$ we have not needed to determine the exact order of the clauses in C_n, as that was irrelevant regard to the output of Algorithm 2 and $\Pi(n)$. We will use the order of the clauses in C_n shown in Figure 4.5. Fixing this particular order determines how the conditional extensions of Algorithm 2 follow each other, which in turn, since $\mathbf{\Pi}$ simulates Algorithm 2, defines how the extensional subconfigurations are nested in

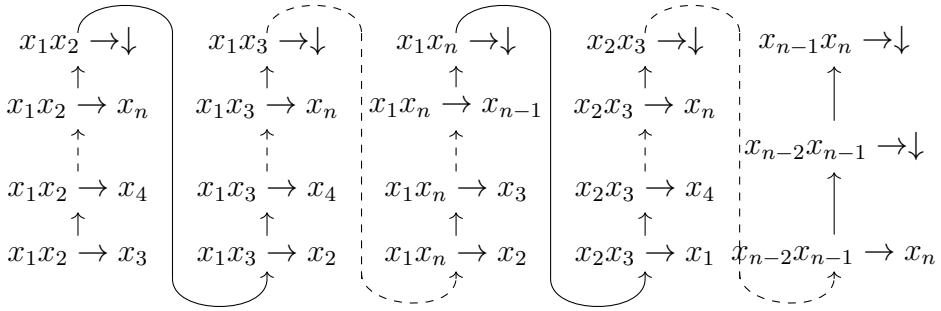


Figure 4.5: *The order of the clauses in C_n.*

each other in μ.

4.5.1 Relabeling the membranes in μ

The membranes of μ can be classified into four groups according to their original labels: clause-membranes, l_d-membranes, l-membranes, and the remaining membranes with labels skin, check, input and g. Each group can be divided into subgroups. To define these subgroups, we define first the *position* of a membrane s, denoted by $pos(s)$, in a linearly nested membrane substructure S as follows. $pos(s) = 1$ if s is the innermost membrane of S, otherwise $pos(s) = pos(s') + 1$, where s' is the child membrane of s. Consider an extensional subconfiguration $ext^i_{\mathcal{C}_j}$ of μ where $\mathcal{C}_j = xy \to z$. The four subgroups of the clause-membranes are defined by the number of comas their labels contain (see Figure 4.2). Similarly, the four subgroups of the l_d-membranes are defined by the position of an l_d-membrane in the substructure $[[[[\ ]_{l_d}]_{l_d}]_{l_d}]_{l_d}$.

It is a bit more difficult to define the appropriate subgroups of the l-membranes, there may be nine in this case. Recall that in $ext^1_{\mathcal{C}_1}$ there are four l-membranes in an l-structure. Moreover, the l-structures of an extensional subconfiguration $ext \neq ext^1_{\mathcal{C}_1}$ contain nine more l-membranes than the l-structures of the extensional subconfiguration right below ext. Thus, in $ext^i_{\mathcal{C}_j}$, there are $\overbrace{9 + \ldots + 9}^{(i-1)m+(j-1)}+4$ l-membranes in an l-structure. Hence, an l-structure can be divided into $(i-1)m+j$ *l-substructures*, where each l-substructure contains nine l-membranes except the innermost one which contains four. A subgroup of l-membranes contains all the l-membranes having the same position in such an l-substructure. Finally, the four subgroups of the remaining membranes with labels skin, check, input and g are defined simply by the four membrane labels.

Now, we modify the labels of the membranes of $ext^i_{\mathcal{C}_j}$ as follows (Figure 4.6 shows $ext^i_{\mathcal{C}_j}$ in the initial membrane configuration after these label modifications). The subscript of the labels of the clause membranes are extended by i and the labels of the

l_d-membranes and the l-membranes are extended by upper indices that indicate the subgroup they belong to. Furthermore, the labels of the l_d-membranes are extended by upper indices containing the parameters i and $xy \to z$, respectively. Let $ext^{i'}_{\mathcal{C}_{j'}}$ be the extensional subconfiguration right below $ext^{i}_{\mathcal{C}_j}$ where $\mathcal{C}_{j'} = x'y' \to z'$. The labels of the l-membranes placed in the outermost l-substructures of $ext^{i}_{\mathcal{C}_j}$ are further extended by subscripts containing i and $xy \to z$. The labels of the l-membranes placed in the l-substructures right below the outermost ones are extended by subscripts containing the parameters i' and $x'y' \to z'$, and this label extension goes on this way to the innermost l-substructures. Recall that $\mathcal{C}_1 = x_1x_2 \to x_3$. The innermost l-substructures contain four l-membranes, respectively, whose labels are extended by subscripts containing 1 and $x_1x_2 \to x_3$. Furthermore, since the label of the innermost l-membrane in all l-structures is the same, it is not necessary to modify the labels of the g-membranes. In summary, after the label modifications, each label in $ext^{i}_{\mathcal{C}_j}$ contains parameters $p_1 \in [n-1]$ and $p_2p_3 \to p_4 \in C_n$, except the g-membranes. We call these parameters *ext-parameters* in what follows. Moreover, we denote the set of these modified labels by H^{mod}.

Finally, we also need to modify the rules in R by replacing the membrane labels occurring in them with their counterparts in H^{mod}.

4.5.2 Encoding of the inputs, the labels and objects of $\Pi(n)$

As Turing machines are uniform computational devices in the sense that the same computational device is used for all possible input lengths, we need to give a reasonable encoding of the labels and objects appearing in words of $L_{\mathbf{\Pi}}$ and L_{H3SN}. We describe a binary encoding of the labels first. These binary strings are of $6+(4\lceil \log n \rceil+1)$ length. We denote the set of these strings by H^{bin}, thus $H^{bin} \subseteq \{0,1\}^{4\lceil \log n \rceil+7}$. The first two bits of a string of H^{bin} indicate the group that contains the membrane:

- 00: a clause-membrane,
- 01: an l_d-membrane,
- 10: an l-membrane, and
- 11: a membrane with label skin, check, input or g.

The following four bits of such a string indicate the subgroup which contains the membrane. A subgroup of clause-membranes can be easily indicated by setting exactly one bit of the four to 1 in the appropriate position. A subgroup of l_d-membranes can be indicated in the same way as well as a subgroup of membranes with labels skin, check, input and g. In the case of the l-membranes, each subgroup is indicated by a number from 1 to 9 whose binary form requires up to four bits. Consequently, a subgroup of l-membranes can be indicated by one of the strings $0001, 0010, \ldots, 1001$.

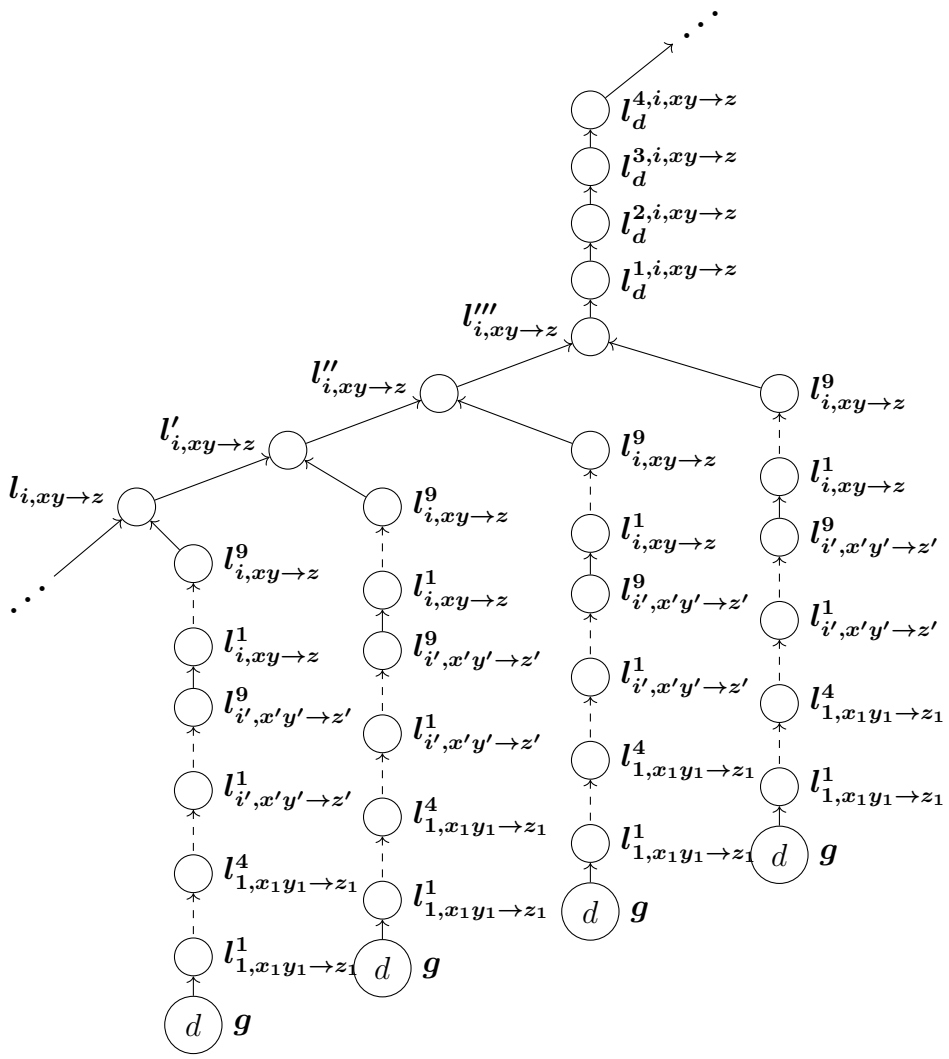


Figure 4.6: *The subconfiguration $ext^i_{\mathcal{C}_{xy\to z}}$ of μ after the modification of the membrane labels. The indices $i', x'y' \to z'$ and $1, x_1y_1 \to z_1$ are the parameters of the extensional subconfiguration right below $ext^i_{\mathcal{C}_{xy\to z}}$ and the innermost extensional subconfiguration, respectively.*

In $ext^i_{\mathcal{C}_j}$, the labels of the clause-membranes, the l_d-membranes and the l-membranes contain some sort of ext-parameters $p_1 \in [n-1]$ and $p_2p_3 \to p_4 \in C_n$. The remaining $4\lceil \log n \rceil + 1$ bits are used to encode these ext-parameters. The encoding of p_1 and $p_2, p_3 \in Var_n$ requires $\lceil \log n \rceil$ bits, respectively, to encode $p_4 \in Var_n \cup \{\downarrow\}$, at most $\lceil \log n \rceil + 1$ bits are needed. We can assume that the encoding of p_1 is simply the number $p_1 - 1$ in binary form, the encoding of a variable x_k with $k \in [n]$ is $k-1$ in binary form, and the binary string that represents $\downarrow$ is the binary form of n. The labels skin, check, input and g do not posses ext-parameters, hence the last $4\lceil \log n \rceil + 1$ bits of their encodings do not contain any information. Therefore, we can assume that all

of these bits are set to 0 in this case. We denote the binary encoding of a label ℓ by $enc(\ell)$, and the structure of $enc(\ell)$ can be seen in Figure 4.7.

$$\underbrace{b_1b_2}_{\text{group}}\ \underbrace{b_3b_4b_5b_6}_{\text{subgroup}}\ \underbrace{b_7\dots b_q}_{i}\ \underbrace{b_{q+1}\dots b_r}_{x}\ \underbrace{b_{r+1}\dots b_s}_{y}\ \underbrace{b_{s+1}\dots b_{4\lceil\log n\rceil+7}}_{z}$$

Figure 4.7: *The structure of a string of H^{bin}.*

We also define an encoding of the objects which we will utilize later. First, we specify four groups of objects: objects of the form $c_{xy\to z}$, objects of the forms v_x and $\overline{v}_x$, and all the remaining objects $(d, \hat{d}, yes, no)$. We assign a binary string of $2+3\lceil\log n\rceil+1$ length to each object $o \in O$. The first two bits of this string represent the group that o is contained by:

- 00: $o = c_{xy\to z}$ for some $xy \to z \in C_n$,
- 01: $o = v_x$ for some $x \in Var_n \cup \{\downarrow\}$,
- 10: $o = \overline{v}_x$ for some $x \in Var_n \cup \{\downarrow\}$,
- 11: $o \in \{d, \hat{d}, yes, no\}$.

If $o = c_{xy\to z}$ then the remaining $3\lceil\log n\rceil+1$ bits are used to encode x, y and z. If $o = v_x$ or $o = \overline{v}_x$ then the last $\lceil\log n\rceil+1$ bits encode x in both cases. We can assume that all the other bits are set to 0. If $o \in \{d, \hat{d}, yes, no\}$ then the third and the fourth bits encode these four possibilities. Again, we can assume that all the other bits are set to 0. The binary encoding of o is denoted by $enc(o)$.

Before giving an appropriate encoding of input formulas, for technical reasons, we modify slightly the instances of the problem H3SN. Let φ be an instance of H3SN. From now on, each clause of φ is either a positive unit clause, a clause containing exactly two negative literals and at most one positive literal, or a clause having the form $x\vee\uparrow$ for some $x \in Var_n$, where $\uparrow$ denotes the constant $true$ value. Moreover, for every $x \in Var_n$, either x is a unit clause of φ or $x\vee\uparrow$ is a clause of φ, but not both. For example, the formula $\varphi_{ex} = x_1x_2 \to\downarrow \wedge x_1 \wedge x_3 \wedge (x_2\vee\uparrow)$ with variables in Var_3 is an instance of this variant of H3SN.

Notice that the encoding cod of an input formula φ yields the same multiset as before. However, now $cod(\varphi)$ contains an object $\overline{v}_x \neq \overline{v}_\downarrow$ if and only if $x\vee\uparrow$ is a clause of φ. Since for any truth assignment $\mathcal{I}$, $\mathcal{I}(x\vee\uparrow) = true$, the satisfiability of φ depends only on those clauses which are positive unit clauses or contain two negative literals. Therefore, H3SN is **P**-complete in this modified form as well.

Now, we present a binary encoding of φ which will be processed by a **DLOGTIME** Turing machine. Each non-unit clause of the form $xy \to z$ is encoded

by a string $\#_{nu}\, bin_{nu}$ where $bin_{nu} \in \{0,1\}^{3\lceil \log n \rceil + 1}$ is the concatenation of the encodings of x, y and z. The encoding of the clauses of the forms x and $y \vee \uparrow$ are the strings $\#_u\, bin_u$ and $\#_{\overline{u}}\, bin_{\overline{u}}$, respectively, where $bin_u, bin_{\overline{u}} \in \{0,1\}^{\lceil \log n \rceil}$ are the encoding of x and y, respectively. The binary encoding of φ is the concatenation of the encoding of its clauses. The characters $\#_{nu}, \#_u$ and $\#_{\overline{u}}$ will be used to retrieve information about the form of the encoded clause and the position where its encoding starts. We denote the binary encoding of φ by $enc_{bin}(\varphi)$. The binary encoding of the formula $\varphi_{ex} = x_1 x_2 \rightarrow \downarrow \wedge x_1 \wedge x_3 \wedge (x_2 \vee \uparrow)$ is $enc_{bin}(\varphi_{ex}) = \#_{nu}\, 00\, 01\, 011\ \#_u\, 00\ \#_u\, 10\ \#_{\overline{u}}\, 01$.

Finally, for technical reasons, we modify the input multiset $cod(\varphi)$ in two steps as follows. First, we consider an ordering of objects in $cod(\varphi)$: the position of an object in $cod(\varphi) - \{\overline{v}_\downarrow\}$ is equal to the position of that clause in φ which the object represents, furthermore, the last object of $cod(\varphi)$ is $\overline{v}_\downarrow$. For example, $cod(\varphi_{ex}) = c_{x_1 x_2 \rightarrow \downarrow} v_{x_1} v_{x_3} \overline{v}_{x_2}\, \overline{v}_\downarrow$. Next, we extend $cod(\varphi)$ the following way: each object of the form $c_{xy \rightarrow z}$ is followed by $3\lceil \log n \rceil + 1$ $\hat{d}$-copies, moreover, all objects of the form v_x or $\overline{v}_x$, where $x \neq \downarrow$, are followed by $\lceil \log n \rceil$ $\hat{d}$-copies. Recall that $\hat{d}$ is a so-called dummy object, that is, the appearance of its copies in any membrane does not have any effect on the computation of $\Pi(n)$. Due to this modification, we need to extend $\Sigma(n)$ as well by adding $\hat{d}$ to it. After these modifications, we have $cod(\varphi_{ex}) = c_{x_1 x_2 \rightarrow \downarrow} \hat{d}\hat{d}\hat{d}\hat{d}\hat{d}\hat{d}\hat{d}\hat{d}\, v_{x_1} \hat{d}\hat{d}\, v_{x_3} \hat{d}\hat{d}\, \overline{v}_{x_2} \hat{d}\hat{d}\, \overline{v}_\downarrow$.

Observe that

1. $|cod(\varphi)| = O(n+m) = O(n^3)$,
2. $|H^{mod}| = O(nm + k_{n-1,m}) = O(n^4)$,
3. $|R| = |R_1| + |R_2| + |R_3| = O(nm + k_{n-1,m}) = O(n^4)$, and
4. the size of μ is $O(nmk_{n-1,m}) = O(n^8)$.

Therefore, both the size of $cod(\varphi)$ and the size of $\Pi(n)$ are polynomial in n. This is a basic necessity for the proper behavior of a **DLOGTIME** Turing machine that recognizes $L_{\mathbf{\Pi}} \cup L_{\text{H3SN}}$.

4.5.3 Recognizing $L_{\mathbf{\Pi}} \cup L_{\text{H3SN}}$ by a DLOGTIME Turing machine

We show that there exists a **DLOGTIME** Turing machine M that recognizes $L_{\mathbf{\Pi}} \cup L_{\text{H3SN}}$. For this, we give an explanation of how only the appropriate words describing the various properties of $\Pi(n)$ (discussed in Definition 4.2.1) can be accepted by M. First, we assume that the input formula, the objects, and the membrane labels in the input of M are given in their encoded form, which encoding was defined above, and the numbers are given in their binary form. Recall that **DLOGTIME** Turing machines are able to determine the length of their input (consequently, the value of n from 1^n), compute sums and differences, multiply a number by a constant and

calculate logarithm. It is also able to check the equality of two strings of $O(\log n)$ length, which is useful when comparing two binary numbers. Therefore, M is able to determine whether some i and $xy \to z$ are possible ext-parameters according to Figure 4.5. Thus, it is **DLOGTIME**-computable whether a string represents a valid object or a label.

Membrane structure. Let $h_1, h_2 \in H^{mod}$. To describe how to set up M so that it accepts the word $\langle 1, 1^n, h_1, h_2 \rangle$ if and only if $\langle 1, 1^n, h_1, h_2 \rangle \in L^n_{\mathbf{\Pi}}$, we distinguish the main cases according to the choice of h_1 and h_2. In each case, we fix that $i \in [n-1]$ and $xy \to z \in C_n$.

- $h_1 = l_d^{k,i,xy\to z}$ and $h_2 = l_d^{k+1,i,xy\to z}$ ($k \in [3]$):

  In this case, M checks whether those bits of $enc(h_1)$ and $enc(h_2)$ that encode the membrane group and the ext-parameters are the same. It also checks whether the difference of the binary numbers in $enc(h_1)$ and $enc(h_2)$ that encode the membrane subgroups is one.

- $h_1 = l'''_{i,xy\to z}$ and $h_2 = l_d^{1,i,xy\to z}$:

  Again, M checks if the substring encoding the ext-parameters of h_1 and h_2 are the same. Then, it checks whether the first six bits of $enc(h_1)$ and $enc(h_2)$ are $00\ 0001$ and $01\ 1000$, respectively.

- $h_1 = l^9_{i,xy\to z}$ and $h_2 = l^u_{i,xy\to z}$ ($u \in \{\varepsilon, ', '', '''\}$):

  This is very similar to the previous case. The first six bits of $enc(h_1)$ must be $10\ 1001$, and first six bits of $enc(h_2)$ must be $00\ 1000$, $00\ 0100$, $00\ 0010$ or $00\ 0001$, depending on u. Moreover, the substrings encoding their ext-parameters must be the same.

- $h_1 = l''_{i,xy\to z}$ and $h_2 = l'''_{i,xy\to z}$:

  The first six bits of $enc(h_1)$ and $enc(h_2)$ must be $00\ 0010$ and $00\ 0001$, respectively. The remaining bits must be the same. For the pairs of labels $l'_{i,xy\to z}, l''_{i,xy\to z}$ and $l_{i,xy\to z}, l'_{i,xy\to z}$, similar conditions must be checked by M.

- $h_1 = l^k_{i,xy\to z}$ and $h_2 = l^{k+1}_{i,xy\to z}$ ($k \in [8], i \neq 1$ and $xy \to z \neq x_1y_1 \to z_1$):

  The first two bits of $enc(h_1)$ and $enc(h_2)$ must be the same along with the substrings encoding the ext-parameters. M also needs to check if the binary numbers representing the subgroups are less than 9 and their difference is one.

- $h_1 = l^9_{i',x'y'\to z'}$ and $h_2 = l^1_{i,xy\to z}$ ($i' \in [n-1], x'y' \to z' \in C_n$, $ext^{i'}_{x'y'\to z'} \neq ext^1_{x_1x_2\to x_3}$ is right below $ext^i_{xy\to z}$):

  Here, we utilize the fixed order on the elements of C_n and the complex labeling we introduced for the l-membranes. Since h_1 and h_2 must have different ext-parameters, we have two additional cases to investigate. If $i = i'$ then $x'y' \to z'$ is followed by $xy \to z$ in C_n. Otherwise, if $i - i' = 1$ then $x'y' \to z' = x_{n-1}x_n \to\downarrow, xy \to z = x_1x_2 \to x_3$ must hold. Without going into details, one can see that M can be set up so that it is able to determine whether $x'y' \to z'$ is followed by $xy \to z$ in C_n. Indeed, M must compare only three pairs of variables, each of them encoded using $O(\log n)$ bits. The special case, when $h_1 = l^4_{1,x_1x_2\to x_3}, h_2 = l^1_{1,x_1x_2\to x_4}$, can be checked similarly.

- $h_1 = g$ and $h_2 = l^1_{1,x_1x_2\to x_3}$:

  Since the parent membrane of a g-membrane is always a membrane labeled with $l^1_{1,x_1x_2\to x_3}$, if $h_1 = g$ then M only needs to check if $h_2 = l^1_{1,x_1x_2\to x_3}$.

- $h_1 = l^{4,i',x'y'\to z'}_d$ and $h_2 = l_{i,xy\to z}$ ($i, i' \in [n-1], xy \to z, x'y' \to z' \in C_n$, $ext^{i'}_{x'y'\to z'}$ is right below $ext^i_{xy\to z} \neq ext^1_{x_1x_2\to x_3}$):

  By processing the ext-parameters of h_1 and h_2, M is able to check whether $ext^{i'}_{x'y'\to z'}$ is right below $ext^i_{xy\to z}$, whether the membrane with label h_1 is the outermost l_d-membrane of $ext^{i'}_{x'y'\to z'}$, and also that whether the membrane with label h_2 is the innermost clause-membrane of $ext^i_{xy\to z}$.

- $h_1 = \mathsf{check}$ and $h_2 = \mathsf{skin}$, $h_1 = l^{4,n-1,x_{n-1}x_n\to\downarrow}_d$ and $h_2 = \mathsf{check}$, $h_1 = \mathsf{input}$ and $h_2 = l_{1,x_1x_2\to x_3}$:

  Without going into details one can see that these cases can be handled easily by M as well.

Clearly, as M is able to handle all the cases listed above, it is able to reject any other word of the form $\langle 1, 1^n, h_1, h_2\rangle$.

Next, we describe how to set up M so that it accepts a word of the form $\langle 2, 1^n, h, t\rangle$ if and only if $\langle 2, 1^n, h, t\rangle \in L^n_{\mathbf{\Pi}}$. Notice that each clause membrane has exactly two child membranes, each g-membrane is an elementary membrane as well as the input-membrane, and any other membrane has exactly one child membrane. Hence, if $t \notin \{0, 1, 2\}$ then M rejects $\langle 2, 1^n, h, t\rangle$. On the other hand, $\langle 2, 1^n, h, t\rangle$ must be accepted only in a constant number of cases. To determine these cases, M needs to extract from $enc(h)$ and process only the substrings encoding the membrane group and the membrane subgroup.

Initial multisets. Recall that the initial multiset in each g-membrane and in the input-membrane is d. For any other membrane s, $\omega(s) = \varepsilon$. Consequently, we have $\langle 3, 1^n, g, 1, d\rangle, \langle 3, 1^n, \mathsf{input}, 1, d\rangle \in L^n_{\mathbf{\Pi}}$, and any other word of the form $\langle 3, 1^n, h, i, a\rangle$ is not in $L^n_{\mathbf{\Pi}}$. Clearly, M can decide in log-time by checking $enc(h), i$ and $enc(a)$ whether to accept its input word concerning feature 3 or not.

Dissolution rules. Although the rules of $\Pi(n)$ are now prepared to handle membranes with labels in H^{mod}, to avoid cumbersome notation in what follows, we keep referring to the original rules of $\Pi(n)$ (see Table 4.1). First, consider those rules whose left-hand side contains the object d. We can easily set up M so that it accepts the word $\langle 4, 1^n, \mathsf{input}, d, \hat{d}\rangle$, as well as the words of the forms $\langle 4, 1^n, h_1, d, \hat{d}\rangle$ and $\langle 4, 1^n, h_2, d, d\rangle$ where h_1 is the label of an l_d-membrane and h_2 is the label of a clause-membrane or an l-membrane. The ext-parameters of h_1 and h_2 are irrelevant in these cases. On the other hand, it rejects any other word $\langle 4, 1^n, h, d, b\rangle$ with $h \in H^{mod}, b \in O$. Moreover, M can be set up so that it accepts $\langle 4, 1^n, \mathsf{check}, \overline{v}_{\downarrow}, yes\rangle$ and $\langle 4, 1^n, \mathsf{check}, v_{\downarrow}, no\rangle$, but no other words that are related to the check-membrane.

Consider now a word v of the form $\langle 4, 1^n, h, a, b\rangle$. This word corresponds to a rule of the form $[c_{xy\to z}]_{l_{xy\to z}} \to c_{xy\to z}$ if and only if the following conditions hold: $a = b$, $enc(a)$ starts with 00 (i.e., a is of the form $c_{xy\to z}$), $enc(h)$ starts with $00\,1000$ (i.e., h is the label of an innermost clause-membrane), and the last $3\lceil \log n\rceil + 1$ bits of $enc(a)$ and $enc(h)$ are the same. Clearly, M can decide these conditions.

The method is a bit different when v corresponds to a rule of the form $[v_x]_{l'_{xy\to z}} \to v_x$. In this case, the following conditions have to be checked by M: $a = b$, $enc(a)$ starts with 01 and $enc(h)$ starts with $00\,0100$. Moreover, the binary substring encoding the only variable in $enc(a)$ and the substring encoding the first variable of the second ext-parameter in $enc(h)$ have to be the same. This can be checked by M. The remaining cases can be handled similarly.

Input multiset. We set up M so that it accepts a word $\langle 5, \varphi, k, a\rangle$ if and only if $\langle 5, \varphi, k, a\rangle \in L_{\text{H3SN}}$ as follows. First of all, if $k > |enc(\varphi)_{bin}| + 1$ then M rejects the word. Otherwise, it extracts the kth character of $enc_{bin}(\varphi)$. According to this kth character α we have the following cases.

- $\alpha = \#_{nu}$: M checks whether $enc(a)$ starts with 00 and then compares the substring of $3\lceil \log n\rceil + 1$ length starting at the $(k+1)$th position in $enc_{bin}(\varphi)$ and the last $3\lceil \log n\rceil + 1$ bits of $enc(a)$.
- $\alpha = \#_u$: M checks whether $enc(a)$ starts with 01 and then checks whether the substring of $\lceil \log n\rceil$ length starting at the $(k+1)$th position in $enc_{bin}(\varphi)$ is equal to the last $\lceil \log n\rceil$ bits of $enc(a)$.
- $\alpha = \#_{\overline{u}}$: M checks whether $enc(a)$ starts with 10 and then checks whether the substring of $\lceil \log n\rceil$ length starting at the $(k+1)$th position in $enc_{bin}(\varphi)$ is equal to the last $\lceil \log n\rceil$ bits of $enc(a)$.

- $\alpha \in \{0, 1\}$: M checks whether $enc(a)$ encodes the object $\hat{d}$.

Lastly, if $k = |enc_{bin}(\varphi)| + 1$ then $a = \overline{v}_{\downarrow}$ must hold.

With this, we have finished the description of the **DLOGTIME** Turing machine M and showed that it recognizes $L_{\mathbf{\Pi}} \cup L_{\text{H3SN}}$. Consequently, $\mathbf{\Pi}$ is a **DLOGTIME**-uniform family of dissolution P systems capable of solving the HORN3SATNORM problem in polynomial time. Herewith, we have finished the proof of Theorem 2. □

4.6 Concluding remarks

In this chapter, we have shown that **DLOGTIME**-uniform families of dissolution P systems, a variant of polarizationless P systems employing only dissolution rules, can solve the **P**-complete HORN3SATNORM problem efficiently. Since dissolution P systems running in polynomial time are known to be able to solve only problems in **P**, our result gives a new characterization of the complexity class **P** by families of rather restricted polarizationless P systems under very tight uniformity conditions.

Many efficient solutions are known for problems in **P** or for the simulation of polynomial-time deterministic Turing machines by P systems with active membranes. However, our result proves that a very limited context-sensitivity hidden in the membrane structure, together with object evolution provided by dissolution rules, is sufficient to solve **P**-complete problems with polarizationless P systems.

Due to Păun's conjecture, polynomial-time polarizationless P systems with active membranes presumably can solve only problems in **P** if non-elementary membrane division is not allowed. If this were the case, our result would have an interesting consequence: the computational power of dissolution P systems running in polynomial time remains the same even if we allow these P systems to use evolution, communication, and elementary membrane division rules, too.

Chapter 5

On the power of weak non-elementary membrane division rules

In this chapter, we turn our attention to polarizationless P systems with active membranes using non-elementary membrane division rules. However, the division rules we consider here are not of the form of those we mentioned in the Introduction of the thesis. We deal with the so-called weak non-elementary rules that are generalizations of the classical elementary membrane division rules. We will describe how the application of these rules works. Then, we show that polarizationless P systems with active membranes are able to solve all problems in $\mathbf{PSPACE}$ even if they use only in-communication rules, dissolution rules and weak non-elementary membrane division rules.

5.1 Introduction

As we mentioned in Chapter 1, it is well known [9, 50, 52] that P systems with active membranes can solve $\mathbf{PSPACE}$-complete problems in polynomial time if they can use non-elementary membrane division rules (see Page 8 for syntax of these rules). Following the literature, we call these rules *strong non-elementary membrane division rules* in this chapter. We note here that in [28] the authors presented a characterization of $\mathbf{PSPACE}$ by P systems employing no non-elementary membrane division rules, but the P systems used there were non-confluent (i.e., strongly nondeterministic).

In [58], it was shown that without any membrane division rules, deterministic P systems with active membranes can solve in polynomial time only problems in $\mathbf{P}$. It is natural to ask what components other than membrane division rules are necessary for these P systems to solve computationally hard problems. As we discussed, according

to Păun's conjecture, P systems without non-elementary membrane division rules can solve only problems in P if we remove the polarizations of the membranes [44].

Polarizations, on the other hand, are known to be not necessary to solve computationally hard problems when the use of non-elementary membrane division rules is allowed. We recall that the **PSPACE**-complete QSAT problem (QSAT is the problem of deciding if a fully quantified Boolean formula is true or not) was solved efficiently in [11] without using polarizations. In fact, the construction presented in [11] does not employ in-communication rules either, and out-communication rules are used only in the last step of the computation to send the answer of the P system out to the environment. Moreover, the employed strong non-elementary membrane division rules are also restricted as they have the form $[[\ ]_{h_1}[\ ]_{h_1}]_{h_0} \rightarrow [[\ ]_{h_1}]_{h_0}[[\ ]_{h_1}]_{h_0}$, that is, the inner membranes on the left-hand side of these rules have the same labels.

We also pointed out that according to [19], dissolution rules are necessary to solve problems beyond **P**, thus, to solve QSAT. However, to the best of the author's knowledge, it is still open whether evolution rules are necessary, too. Notice that in [11], the evolution rules had an essential role in selecting those clauses of the input formula that are satisfied by a truth assignment of the variables. Although dissolution rules and elementary membrane division rules can evolve objects, it is not clear how they could be used to take over the roles of evolution rules in [11].

In [24], the Q3SAT problem, a **PSPACE**-complete restriction of QSAT was solved by polynomial-time polarizationless P systems using no evolution and communication rules. However, this solution was achieved by a semi-uniform family of P systems. For any decision problem D, a family $\mathbf{\Pi} = \{\Pi(x) \mid x \text{ is an instance of } D\}$ of P systems solving D is semi-uniform if there is an appropriate Turing machine that computes a reasonable representation of $\Pi(x)$ whenever it is started with x on its input tape. Clearly, uniformity conditions are tighter than that of semi-uniformity. Moreover, in [24], the presented P systems used not only strong non-elementary membrane division rules but so-called weak division rules for non-elementary membranes. These rules are in fact generalizations of the well-known elementary membrane division rules to non-elementary membranes and thus have the form $[a]_h^{\alpha_1} \rightarrow [b]_h^{\alpha_2}[c]_h^{\alpha_3}$, where $\alpha_1, \alpha_2, \alpha_3$ are polarizations (see also [8, 57] where the space complexity of P systems employing weak non-elementary membrane division rules is discussed).

Monodirectional P systems were introduced and investigated in [26]. These are P systems with active membranes that have no in-communication rules and have weak non-elementary membrane division rules instead of the strong ones. It is easy to see that in these P systems the information can flow only towards the skin membrane. It turned out in [26] that this syntactical restriction is enough to restrict the computational power of these P systems. More precisely, it was shown in [26] that monodirectional P systems working in polynomial time characterize the class $\mathbf{P}^{\mathbf{NP}}$, that is, the class of problems solved by polynomial-time Turing machines with **NP**

oracles. Interestingly, it turned out also that removing weak non-elementary membrane division rules or dissolution rules does not affect the computational power of these P systems. On the other hand, if both of these rules are removed, then the P systems characterize only $\mathbf{P}_{\|}^{\mathbf{NP}}$. For recent results on the relation of classical complexity classes beyond $\mathbf{NP}$ and classes of problems solvable by various families of P systems with active membranes, we recommend again [51].

In this chapter, we show that weak non-elementary membrane division is enough to efficiently solve $\mathbf{PSPACE}$-complete problems in polarizationless recognizer P systems if dissolution rules and in-communication rules are allowed. More precisely, we show that the QSAT problem can be solved by polynomial-time polarizationless recognizer P systems with active membranes where the applicable types of rules are weak non-elementary membrane division, dissolution, and in-communication. All types of rules used in our P systems are presumably necessary to solve the QSAT problem. Indeed, as we have already discussed, without in-communication, dissolution, or weak non-elementary division these systems can solve presumably less difficult problems due to [26], [19], and [58], respectively (notice that although in [19] the P systems employ strong non-elementary membrane divisions, their result can easily be adopted to those P systems which have weak non-elementary divisions instead of the strong ones).

Our proof resembles the one given in [11]. That solution of QSAT first builds a binary tree of membranes starting from a linearly nested membrane structure. Each elementary membrane in this tree corresponds to a truth assignment of the variables of the input. Our solution builds a similar tree of membranes, but a truth assignment is associated with a set of elementary membranes instead of one. However, in [11] this binary tree is built in a bottom-up manner, that is, the strong non-elementary division rules divide the innermost membrane first and the division is propagated towards the skin membrane. Meanwhile, the mentioned truth assignments are created using elementary membrane division rules, that is, they appear at the leaves of the created binary tree. On the other hand, the weak non-elementary division rules used in this chapter create a binary tree of the membranes in a top-down manner: first, one of the upper membranes of the tree is divided and the division continues towards the elementary membranes. The weak non-elementary divisions are used to create the objects representing the truth assignments as well. Then these objects are sent to the elementary membranes using in-communication rules. This means that in-communication rules play an important role in our result. In fact, as we have discussed above, QSAT cannot be solved without in-communication rules by our P systems unless $\mathbf{PSPACE} = \mathbf{P}^{\mathbf{NP}}$.

The remainder of the chapter is organized as follows. In the next section, we discuss the syntax and semantics of weak division rules for non-elementary membranes. In Section 5.3 we define the QSAT problem, and Algorithm 3 that solves QSAT. Then,

in Section 5.4, we present a $\mathbf{P}$-uniform family of P systems with active membranes that employ only in-communication rules, dissolution rules and weak division rules for non-elementary membranes, and solves QSAT in polynomial time by implementing Algorithm 3. In Section 5.5, we make some concluding remarks.

5.2 Weak division rules for non-elementary membranes

In this chapter, we deal with polarizationless P systems with active membranes but on top of the classical rules we also consider *weak division rules for non-elementary membranes* of the form

$$[a]_h \rightarrow [b]_h[c]_h$$

for some $h \in H$ and $a, b, c \in O$.

This type of rules are generalizations of the classical elementary membrane division rules for non-elementary membranes with the following semantics. Consider a membrane configuration $C = (V, E, L, \omega)$, a weak division rule r for non-elementary membranes, and an edge $(x, y) \in E$ such that x is a subject of r (that is, $L(x) = h$) and $a \in \omega(x)$. Let (V', E') be that subtree of (V, E) which has x as the root. The application of r to x duplicates the subtree (V', E') into two disjoint trees (V'_1, E'_1) and (V'_2, E'_2) with roots x_1 and x_2, respectively, and replaces the edge (x, y) with edges (x_1, y) and (x_2, y). Each node n of V'_1 (resp. of V'_2) is associated with the same label as that of the corresponding node in V'. Moreover, x_1 (resp. x_2) is associated with $(\omega(x) - a) + b$ (resp. with $(\omega(x) - a) + c$), and the rest of the nodes in V'_1 (resp. in V'_2) are associated with the same object multiset as that of the corresponding node in V'. The rules of the P system are applied in the usual bottom-up manner during a computation step. Hence, when a weak division rule is applied to a membrane x then all the rules assigned to the objects in the subtree with root x are assumed to be applied already.

5.3 The QSAT problem.

In this chapter, we are going to construct P systems with active membranes that can decide whether a *Quantified Boolean formula* (QBF) is true or not. QBFs extend propositional logic by allowing universal and existential quantifiers over propositional variables. It is assumed that each variable is under the bound of at most one quantifier. We consider fully quantified QBFs in *prenex normal form*, that is, in the form $F = Q_1 x_1 \ldots Q_n x_n \Phi(x_1, \ldots, x_n)$, where $n \geq 1$, Q_i ($i \in [n]$) is either $\exists$ or $\forall$, and $\Phi(x_1, \ldots, x_n)$ is the quantifier-free part of F indicating that the variables of Φ are in Var_n.

Let $\Phi(x_1, \ldots, x_n)$ be a quantifier-free formula, $i \in [n]$, and, for every $j \in [i]$, $v_j \in \{true, false\}$. Then $\Phi(v_1, \ldots, v_i, x_{i+1}, \ldots, x_n)$ is that formula $\Phi'(x_{i+1}, \ldots, x_n)$ that we get from Φ by replacing, for every $j \in [i]$, the variable x_j by v_j. Clearly, the truth values $true$ and $false$ serve in Φ' as logical constants with the corresponding truth values.

In this chapter, we consider the following **PSPACE**-complete variant of the QSAT problem: the input is a fully quantified QBF

$$F = \exists x_1 \forall x_2 \ldots \exists x_{n-1} \forall x_n \Phi(x_1, \ldots, x_n)$$

in prenex normal form, where $n \geq 2$ and Φ is in CNF. The task is to decide whether F is *true* or *false*. Clearly, F is *true* if and only if the following holds:

$$\begin{aligned}(&\textbf{exists } v_1 \in \{true, false\})(\textbf{for all } v_2 \in \{true, false\}) \ldots \\ &\ldots (\textbf{exists } v_{n-1} \in \{true, false\})(\textbf{for all } v_n \in \{true, false\}) \\ &(\Phi(v_1, \ldots, v_n) \equiv true)\end{aligned} \quad (5.1)$$

We will show that QSAT can be solved by polarizationless P systems with active membranes using only weak non-elementary division, in-communication and dissolution rules. Roughly, our P systems will implement a function Eval given in Algorithm 3 which is, in fact, a variant of the well-known basic method of recursively evaluating Expression (5.1).

Algorithm 3 the decision of QSAT

```
1: function EVAL(Φ(x_i, ..., x_n), i)                    ▷ initially, i = 1
2:   if (i ≤ n) then
3:     if (i mod 2 == 0) then
4:       return EVAL(Φ(true, x_{i+1}, ..., x_n), i + 1) ∧ EVAL(Φ(false, x_{i+1}, ..., x_n), i + 1)
5:     else
6:       return EVAL(Φ(true, x_{i+1}, ..., x_n), i + 1) ∨EVAL(Φ(false, x_{i+1}, ..., x_n), i + 1)
7:     end if
8:   else
9:     for all C ∈ Φ do                                  ▷ no free variables occur in Φ
10:      if true ∉ C then
11:        return false
12:      end if
13:    end for
14:    return true
15:  end if
16: end function
```

5.4 The solution for QSAT

This section is devoted to the proof of the main result of the chapter which can be stated as follows:

Theorem 3. *The* QSAT *problem can be solved in polynomial time by a polynomially uniform family* $\mathbf{\Pi} = \{\Pi(n,m) \mid n \geq 2, m \geq 1\}$ *of polarizationless recognizer P systems with active membranes such that the members of* $\mathbf{\Pi}$ *employ only in-communication rules, dissolution rules and weak division rules for non-elementary membranes.*

For $n \geq 2$ and $m \geq 1$, QSAT(n,m) denotes the set of those instances of QSAT which have m clauses and variables in Var_n. We will use $\Pi(n,m)$ to decide whether the instances of QSAT(n,m) are true or not.

For the remainder of this proof, we fix an instance

$$F = \exists x_1 \forall x_2 \ldots \exists x_{n-1} \forall x_n \Phi(x_1, \ldots, x_n)$$

of QSAT(n,m). Denote C_j ($j \in [m]$) the jth clause of Φ. First, we define the encoding of F, denoted by $cod(F)$, as a subset of $\Sigma(n,m) = \{v_{i,j}, v'_{i,j} \mid i \in [n], j \in [m]\}$ in the following way:

$$cod(F) = \{v_{i,j} \mid x_i \text{ occurs in } C_j\} \cup \{v'_{i,j} \mid \neg x_i \text{ occurs in } C_j\}.$$

We now construct the P system $\Pi(n,m) = (O, H, \mu, R)$ as follows:

- $O = \{t_i, f_i, t'_i, f'_i, \hat{t}_i, \hat{f}_i, \mid i \in [n]\} \cup \Sigma(n,m) \cup \{a, T, s, yes, no\}$,
- $H = \{x_i, x'_i, \alpha_i, \beta_i \mid i \in [n]\} \cup \{C_j \mid j \in [m]\} \cup \{p, l, h, x_{n+1}, \mathsf{skin}, \mathsf{halt}, \mathsf{input}\}$,
- the input membrane and the output membrane are labeled with input and skin, respectively, and
- the initial membrane configuration (μ) and the set of rules (R) are defined below.

The initial membrane configuration $\mu = (V, E, L, \omega)$ can be seen in Figure 5.1. The labels of the membranes are placed beside the nodes, and the multisets associated with the nodes are written in the nodes. Notice that some nodes denote subconfigurations that are explained separately. The root of the membrane structure is the skin-membrane which has an only child with label halt. Below halt there are two subconfigurations called the *working subtree* and the *countdown subtree*, respectively (see Figure 5.1a). The working subtree (Figure 5.1b) will be used to decide whether the input formula F is true. More precisely, this part will produce at least one T-copy in the halt-membrane if and only if F is *true*. Then a T-copy triggers the dissolution of the halt-membrane resulting in the appearance of *yes* in the skin-membrane. The countdown subtree (Figure 5.1c) will be used to produce object s in

the halt-membrane if and only if F is *false*, that is, no T-copy was produced in the halt-membrane by the working subtree. Then s triggers the dissolution of the halt-membrane resulting in the appearance of *no* in the skin-membrane. The countdown subtree has $7n+2m+5$ linearly nested l-membranes and an elementary h-membrane in the innermost l-membrane. Initially, all l-membranes are empty, and for the only h-membrane x, $\omega(x) = s$.

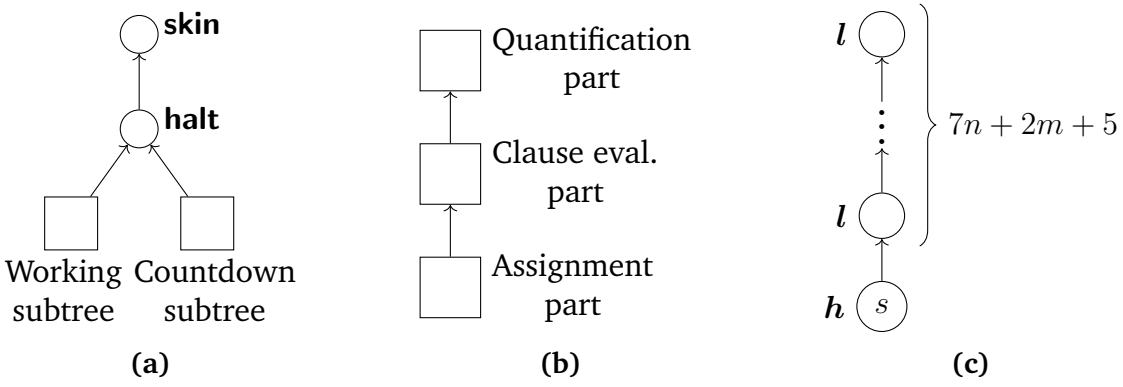


Figure 5.1: *The initial membrane configuration (a), the working subtree (b) and the countdown subtree (c). The rectangles denote subconfigurations and the circles denote membranes.*

The working subtree is divided into three parts: the *quantification*, the *clause evaluation*, and the *assignment* parts. We now describe the structure of these parts and give a brief, general description of their role in the whole computation. A more detailed explanation about their operation will be given later.

The quantification part will be used to create exponentially many subtrees of the membrane structure, each of them containing objects that encode a possible truth assignment of the variables of F. In this part, we have membranes with unique labels in $\{x_i, x'_i \mid i \in [n]\} \cup \{x_{n+1}\}$ and several other membranes with the label p. The membrane structure of the quantification part can be seen in Figure 5.2a. It is a single path with a root that is labeled with x_1, and a leaf that is the innermost p-membrane. Figure 5.2a shows the labels of the other nodes in this part. Notice that there is a regularity in the labeling below the x_2-membrane: for nodes x, y, z with $(z, y), (y, x) \in E$ and $i \in [3, n]$, if $L(x) = x_i$, then $L(y) = p$ and $L(z) = x'_i$. Labels in $\{x_i, x'_i \mid i \in [n]\}$ refer to the corresponding variables of F, while the label p refers to the word “prison", because the a-copy which is initially placed in the p-membranes cannot trigger any rule – so it is imprisoned – until another object sets it free by using a dissolution rule. The membranes of the quantification part are associated with initial multisets of objects as follows. Let x be a node of this part. If x is the x_2-membrane or it is a p-membrane, then $\omega(x) = a$, and $\omega(x) = \varepsilon$ otherwise.

The clause evaluation part will be used to check if a truth assignment of the variables satisfies all the clauses. In this part, we have membranes with unique labels in $\{C_1, \ldots, C_m\}$. These labels refer to the corresponding clauses of F. The membrane structure of this part can be seen in (Figure 5.2b). It is a single path with a root labeled with C_1 and descendants of this root labeled with $C_2, \ldots, C_m$, respectively. Initially, these membranes are all empty, that is, for each node x in this part $\omega(x) = \varepsilon$.

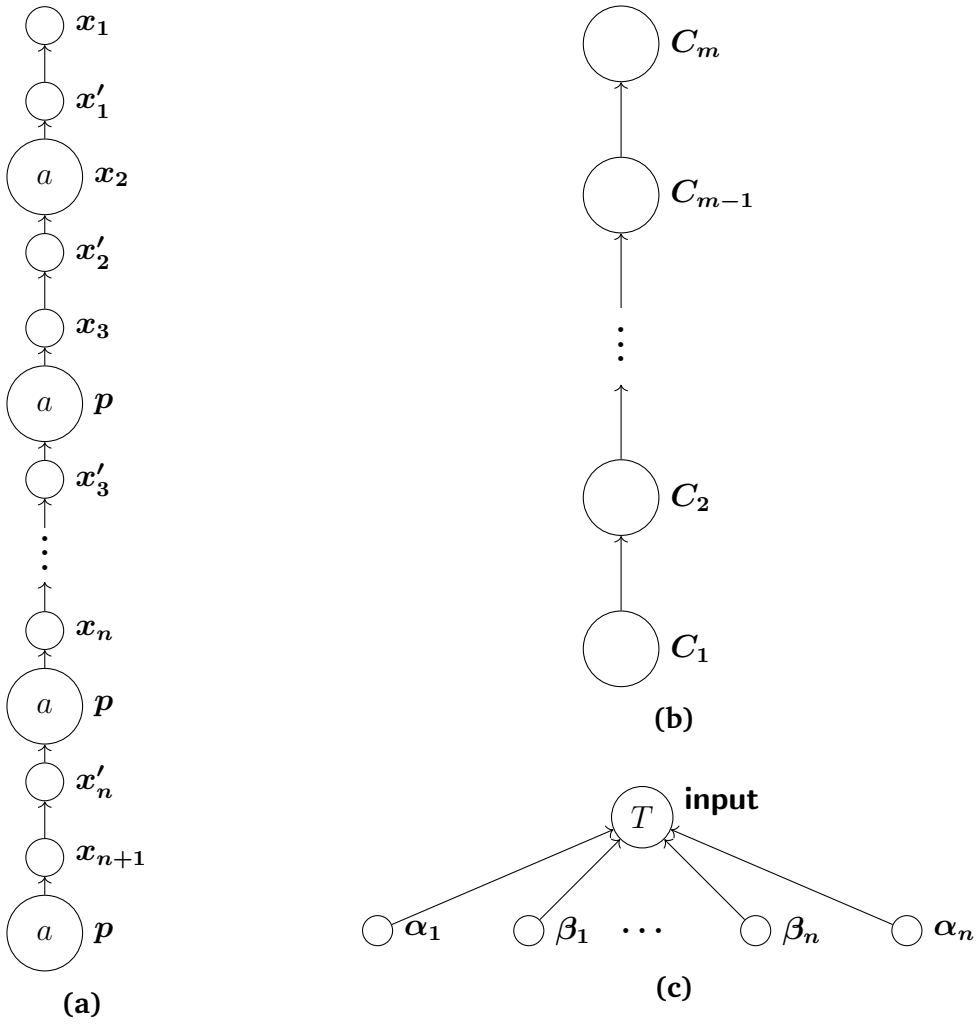


Figure 5.2: *The quantification (a), the clause evaluation (b), and the assignment part (c) of the working subtree of the initial membrane configuration*

The assignment part (Figure 5.2c) will be used to select those objects from the input that represent literals that can make a clause true according to the corresponding truth assignment. This part of the working subtree is a tree with height one. Its root is labeled with input, and below the root there are $2n$ membranes labeled with $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_n$, respectively. Notice that these membranes are the only el-

ementary membranes of the working subtree. Initially, these elementary membranes are empty, while for the root x of the assignment part, $\omega(x) = \{T\}$.

With this, we have finished the description of the initial membrane configuration of $\Pi(n, m)$. Now, we set the root of the assignment part, that is the node x with $L(x) = \mathsf{input}$, to be the input membrane of $\Pi(n, m)$.

Next, we present the rules of R. We group them so that we can explain how the whole computation of $\Pi(n, m)$ works. In particular, the correctness of $\Pi(n, m)$ will also be justified. Therefore, we assume that the input membrane contains not only its initial multiset but also $cod(F)$, the encoding of the input formula F. We distinguish the following two main stages of the computation, the *generation stage* and the *verification stage*.

First, we give the rules applied in the generation stage. Here, the P system generates all the possible truth assignments of the variables of F and sends the objects encoding them down in the membrane structure. This is carried out in the working subtree of the initial membrane structure using several groups of rules presented below.

The following rules divide the x_{i+1}-membranes ($i \in [n]$), that is, duplicate those subtrees of the quantification part which have an x_{i+1}-membrane as the root.

$$\begin{array}{l} \bullet\ [a]_{x_{i+1}} \to [\hat{t}_i]_{x_{i+1}}[\hat{f}_i]_{x_{i+1}}, \quad (i \in [n-1]) \\ \bullet\ [a]_{x_{n+1}} \to [t'_n]_{x_{n+1}}[f'_n]_{x_{n+1}}. \end{array} \qquad (\dagger)$$

After the division, the root of the first new subtree contains either $\hat{t}_i$ with $i \in [n-1]$ or t'_n, while the root of the other one contains either $\hat{f}_i$ with $i \in [n-1]$ or f'_n representing that x_i is set to *true* and *false*, respectively. In fact, the duplication of these subtrees corresponds to the recursive calls in lines 4 and 6 in Algorithm 3 where the first and second calls use Φ with x_i set to $true$ and to $false$, respectively. The reason why we handle case $i = n$ differently by introducing t'_n and f'_n instead of $\hat{t}_n$ and $\hat{f}_n$ will be discussed later.

Initially, only the x_2-membrane can be divided, since the objects that could divide the x_k-membranes with $k \in [3, n+1]$ are imprisoned in the p-membranes. The P system uses the following rules triggered by $\hat{t}_i$ and $\hat{f}_i$, respectively, to dissolve the p-membranes and, in turn, to set objects a free:

$$\left.\begin{array}{l} \bullet\ \hat{t}_i[\ ]_{x'_{i+1}} \to [\hat{t}_i]_{x'_{i+1}}, \\ \bullet\ \hat{t}_i[\ ]_{x_{i+2}} \to [\hat{t}_i]_{x_{i+2}}, \\ \bullet\ \hat{t}_i[\ ]_p \to [\hat{t}_i]_p, \\ \bullet\ [\hat{t}_i]_p \to t'_i, \end{array}\right\} i \in [n-1]$$

$$\left.\begin{array}{l} \bullet\ \hat{f}_i[\ ]_{x'_{i+1}} \to [\hat{f}_i]_{x'_{i+1}}, \\ \bullet\ \hat{f}_i[\ ]_{x_{i+2}} \to [\hat{f}_i]_{x_{i+2}}, \\ \bullet\ \hat{f}_i[\ ]_p \to [\hat{f}_i]_p, \\ \bullet\ [\hat{f}_i]_p \to f'_i, \end{array}\right\} i \in [n-1]$$

Notice that an x_{i+2}-membrane can be divided only after that its ancestor x_{i+1}-membrane is already divided. Indeed, as we have discussed above, to divide an x_{i+2}-membrane an object $\hat{t}_i$ or $\hat{f}_i$ is necessary, and these objects are introduced during the division of an x_{i+1}-membrane.

The evolution of $\hat{t}_i$ to t'_i and $\hat{f}_i$ to f'_i in the above dissolution rules is necessary since we want to send the objects encoding the truth assignments down to the leaves. Without evolving these objects, there would occur undesired nondeterminism: we would have in-communication and dissolution rules triggered by the same objects. Notice, moreover, that the rule $[a]_{x_{n+1}} \to [t'_n]_{x_{n+1}}[f'_n]_{x_{n+1}}$ given in rules (†) needs to introduce no objects marked with $\hat{}$ since when it is applied there are no membranes left to divide. This is why it introduces t'_n and f'_n directly.

The following rules are used to send the objects that represent the truth assignments towards the leaves.

$$\left.\begin{array}{ll} \bullet\ t'_i[\ ]_{x'_k} \to [t'_i]_{x'_k}, & (k \in [i+2, n]) \\ \bullet\ t'_i[\ ]_{x_k} \to [t'_i]_{x_k}, & (k \in [i+3, n+1]) \\ \bullet\ t'_i[\ ]_p \to [t'_i]_p, & \\ & \\ \bullet\ f'_i[\ ]_{x'_k} \to [f'_i]_{x'_k}, & (k \in [i+2, n]) \\ \bullet\ f'_i[\ ]_{x_k} \to [f'_i]_{x_k}, & (k \in [i+3, n+1]) \\ \bullet\ f'_i[\ ]_p \to [f'_i]_p, & \end{array}\right\} i \in [n-2]$$

By applying these rules, objects t'_i and f'_i will reach the innermost p-membranes which are, in fact, the innermost membranes of the quantification part of the working subtree (see Figure 5.2a). The system is set up so that when the x_{i+1}-membranes are divided, the objects t'_j and f'_j with $j < i$ have already passed through the x_{i+1}-membranes. Therefore, these objects are duplicated with the division of the x_{i+1}-membranes. This means that after that, the x_{n+1}-membranes are divided, the working subtree of the P system has 2^n subtrees with x_{n+1}-membranes at the roots (see Figure 5.3). Moreover, each of these subtrees contains a set of objects encoding a possible truth assignment of the variables. Furthermore, all possible truth assignments are represented in these subtrees.

In parallel to the computation carried out in the quantification part, objects in $cod(F)$ are sent from the input membrane to the α_i- and β_i-membranes, respectively,

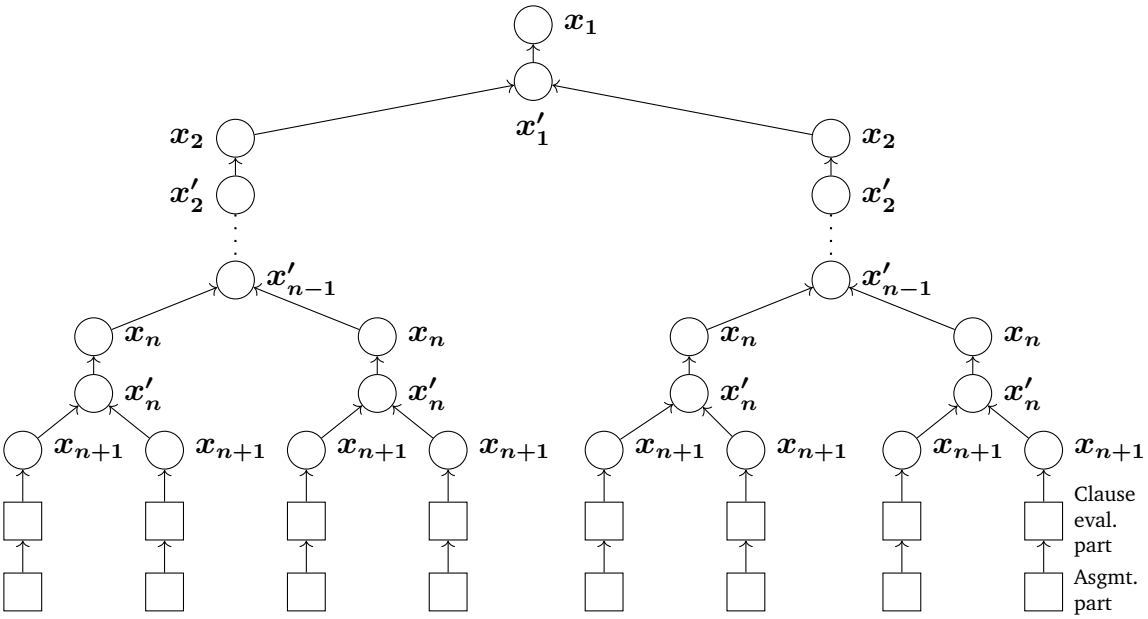


Figure 5.3: *The shape of the working subtree after that all the divisions are done. The rectangles denote subconfigurations and the circles denote membranes.*

using the following rules.

- $v_{i,j}[\ ]_{\alpha_i} \to [v_{i,j}]_{\alpha_i}$,
- $v'_{i,j}[\ ]_{\beta_i} \to [v'_{i,j}]_{\beta_i}$,

$\Big\}\ i \in [n], j \in [m]$

Recall that object $v_{i,j}$ (resp. $v'_{i,j}$) encodes that x_i (resp. $\neg x_i$) occurs in C_j. Therefore, there are at most m objects representing that the ith variable occurs in a clause (with or without negation). This means that in at most m steps all input objects get into the corresponding α_i- or β_i-membranes. The role of sending these objects into these membranes will be discussed later.

The following rules send the objects representing the truth assignments through the clause evaluation part of the working subtree. Notice that the actual evaluation of the clauses does not happen in this part of the computation but in a later stage of it.

- $t'_i[\ ]_{C_j} \to [t'_i]_{C_j}$,
- $t'_i[\ ]_{\mathsf{input}} \to [t'_i]_{\mathsf{input}}$,
- $t'_i[\ ]_{\alpha_i} \to [t'_i]_{\alpha_i}$,
- $f'_i[\ ]_{C_j} \to [f'_i]_{C_j}$,
- $f'_i[\ ]_{\mathsf{input}} \to [f'_i]_{\mathsf{input}}$,
- $f'_i[\ ]_{\beta_i} \to [f'_i]_{\beta_i}$,

$\Big\}\ i \in [n], j \in [m]$

When objects t'_i and f'_i reach the input membrane, they go inside the α_i- and β_i-membranes, respectively. Note that reaching the leaves for t'_1 and f'_1 takes more than m steps (hence, for any of the t'_i, f'_i objects), thus, by the time they do so, all the input objects in $cod(F)$ are already separated into the α_i- and β_i-membranes, respectively. The next rules are used to dissolve certain α_i- and β_i-membranes.

- $t'_i[\ ]_{C_j} \to [t'_i]_{C_j}$,
- $t'_i[\ ]_{\mathsf{input}} \to [t'_i]_{\mathsf{input}}$,
- $t'_i[\ ]_{\alpha_i} \to [t'_i]_{\alpha_i}$,
- $f'_i[\ ]_{C_j} \to [f'_i]_{C_j}$,
- $f'_i[\ ]_{\mathsf{input}} \to [f'_i]_{\mathsf{input}}$,
- $f'_i[\ ]_{\beta_i} \to [f'_i]_{\beta_i}$,

$\Big\}\ i \in [n], j \in [m]$

After an object t'_i (resp. an f'_i) gets into an α_i-membrane (resp. a β_i-membrane), the P system dissolves such membrane using the above rules. Notice that, for each $i \in [n]$ and $x, y \in V$ with $L(x) = \alpha_i$ and $L(y) = \beta_i$, if x and y are siblings, then either x or y is dissolved but not both. Moreover, since t'_n and f'_n are the last objects that reach the leaves, the last membrane that is dissolved in this part of the computation is either an α_n- or a β_n-membrane. After the dissolution, input-membranes contain exactly those objects from $cod(F)$ that correspond to literals that are true under the corresponding evaluation of the variables. Clearly, these literals satisfy the clauses they occur in under this evaluation. During the dissolution of the α_i- and β_i-membranes, objects t'_i and f'_i evolve to t_i and f_i, respectively, which is necessary to avoid undesired nondeterminism in the remaining part of the computation. After dissolving all of the α_i- and β_i-membranes, t_n or f_n dissolves the input-membrane and all objects contained in this membrane, including a T-copy initially placed in it, get to the C_1-membrane.

We now give the rules that are used in the verification stage of the computation. Here, the P system checks first whether a truth assignment satisfies all the clauses, then it combines the information about the different truth assignments appropriately. To evaluate the clauses, it uses the following rules.

- $[v_{i,j}]_{C_j} \to v_{i,j}$,
- $[v'_{i,j}]_{C_j} \to v'_{i,j}$,

$\Big\}\ i \in [n], j \in [m]$

With these rules, $\Pi(n, m)$ in fact implements lines 9–14 of Algorithm 3 by searching, for each clause C of Φ, a literal that occurs in C and is true under the corresponding truth assignment. Let $x \in V$ be an x_{n+1}-membrane and $\mathcal{I}$ denote a truth assignment represented by objects t_i and f_i occurring in the subtree with root x. The subcomputation here starts with an attempt to apply a rule of the form $[v_{i,1}]_{C_1} \to v_{i,1}$ or $[v'_{i,1}]_{C_1} \to v'_{i,1}$. If no object can trigger any of these rules, then the computation halts

in this subtree. Assume now that $v_{i,1}$ occurs in a C_1-membrane y. Then t_i occurs in y, too, and thus $\mathcal{I}$ assigns *true* to x_i. Moreover, by the definition of $cod(F)$, x_i occurs in clause C_1 of F, which yields that $\mathcal{I}$ satisfies C_1. If, on the other hand, $v'_{i,1}$ occurs in y, then f_i should occur here, too. This means that $\mathcal{I}$ assigns *false* to x_i and thus, since in this case $\neg x_i$ should occur in C_1, $\mathcal{I}$ satisfies C_1 in this case, too. That is, the C_1-membrane y is dissolved if and only if $\mathcal{I}$ satisfies C_1. Generalizing this argumentation to the rest of the C_j-membranes one can see that $\Pi(n, m)$ dissolves all the C_j-membranes ($j \in [m]$) if and only if $\mathcal{I}$ satisfies Φ. Thus, at the end of this part of the verification stage (corresponding to the clause evaluation part, see Figure 5.3), the following statement holds.

> (1) An x_{n+1}-membrane either contains no objects, or contains objects t_i and f_i (for each $i \in [n]$, exactly one of t_i or f_i) encoding a satisfying truth assignment of Φ (with t_i and f_i encoding $x_i = \textit{true}$ and $x_i = \textit{false}$ respectively) along with a T-copy.

In the next part of the verification stage, the P system evaluates F using the following rules.

- $[T]_{x_{n+1}} \to T$,
- $[t_i]_{x_i} \to t_i$,
- $[f_i]_{x'_i} \to f_i$,

$\Big\}$ $i \in [n]$ and i is even

and

- $[t_i]_{x_i} \to t_i$,
- $[t_i]_{x'_i} \to t_i$,
- $[f_i]_{x_i} \to f_i$,
- $[f_i]_{x'_i} \to f_i$,

$\Big\}$ $i \in [n]$ and i is odd.

We now describe the computation of $\Pi(n, m)$ using these rules and at the same time show that $\Pi(n, m)$ evaluates F correctly. First, all the x_{n+1}-membranes that contain a T-copy are dissolved. Let $i \in [n]$ and $\tau \in \{t_i, f_i\}$, and let us denote by $v(\tau)$ the truth value represented by τ. To see that $\Pi(n, m)$ evaluates F correctly, we first show the following statement.

> ($\ddagger$) If an x_i-membrane ($i \in [n + 1]$) x is dissolved by $\Pi(n, m)$, then
>
> $$Q_i x_i Q_{i+1} x_{i+1} \dots Q_n x_n \Phi(v(\tau_1), \dots, v(\tau_{i-1}), x_i, x_{i+1}, \dots, x_n) = true,$$
>
> where $Q_i, Q_{i+1}, \dots, Q_n$ are the corresponding quantifiers of F and, for every $j \in [i - 1]$, τ_j is either t_j or f_j according to the objects occurring in x.

Due to the following observation, τ_j in (‡) is well defined.

> (2) For any x_i-membrane ($i \in [n+1]$) x and $j \in [i-1]$, x cannot contain both t_j and f_j during the entire computation of $\Pi(n,m)$.

We now show (‡) by induction on i. If $i = n+1$, then (‡) follows from Statement (1). Assume now that (‡) holds for $1 < i \leq n+1$. We show it for $i-1$. Consider an x_{i-1}-membrane x. We distinguish the following two cases.

1. $i-1$ is even. Looking at the rules above, one can see that to dissolve x, this membrane must contain t_{i-1}. This object can occur in x only if the x'_{i-1}-membrane y below x contains both t_{i-1} and f_{i-1}. The only way to introduce these objects in y is to dissolve some of the x_i-membranes below y. Since by Statement (2) x_i-membranes cannot contain both t_{i-1} and f_{i-1}, to introduce these objects in y, both of the x_i-membranes below y must be dissolved. Then, by the induction hypothesis,

   $$Q_i x_i Q_{i+1} x_{i+1} \dots Q_n x_n \Phi(v(\tau_1), \dots, v(\tau_{i-2}), \nu, x_i, \dots, x_n)$$

   is *true*, for every truth value $\nu \in \{true, false\}$. Therefore,

   $$\forall x_{i-1} Q_i x_i \dots Q_n x_n \Phi(v(\tau_1), \dots, v(\tau_{i-2}), x_{i-1}, x_i, \dots, x_n)$$

   is *true* as well.

2. $i-1$ is odd. In this case, both t_{i-1} and f_{i-1} can dissolve the x_{i-1}-membranes as well as the x'_{i-1}-membranes. Using this and following the argumentation given in the previous case, we can conclude that in this case at least one of the x_i-membranes below x has to be dissolved. Then, by the induction hypothesis, there is ν in $\{true, false\}$ such that

   $$Q_i x_i \dots Q_n x_n \Phi(v(\tau_1), \dots, v(\tau_{i-2}), \nu, x_i, \dots, x_n)$$

   is *true*. Therefore,

   $$\exists x_{i-1} Q_i x_i \dots Q_n x_n \Phi(v(\tau_1), \dots, v(\tau_{i-2}), x_{i-1}, x_i, \dots, x_n)$$

   is *true* as well.

This concludes the proof of (‡). Reversing the argumentation used in this proof, it can be seen that if an x_i-membrane is not dissolved during the computation of $\Pi(n,m)$, then $Q_i x_i \dots Q_n x_n \Phi(v(\tau_1), \dots, v(\tau_{i-1}), x_i, x_{i+1}, \dots, x_n)$ must be $false$. Consequently,

> ($\star$) an x_i-membrane ($i \in [n+1]$) x is dissolved by $\Pi(n,m)$ if and only if $Q_i x_i Q_{i+1} x_{i+1} \dots Q_n x_n \Phi(v(\tau_1), \dots, v(\tau_{i-1}), x_i, x_{i+1}, \dots, x_n)$ is $true$.

Using $(\star)$, it can be seen that, for every $i \in [n]$, if i is even (resp. odd), then the return value in Line 4 (resp. in Line 6) of Algorithm 3 is *true* if and only if the corresponding x_i-membrane is dissolved. Consequently, F is *true* if and only if the x_1-membrane is dissolved. Therefore, a T-copy can reach the halt-membrane if and only if F is *true*.

Using the following, final set of rules, $\Pi(n, m)$ introduces the answer of the P system in the skin-membrane.

- $[s]_h \to s$,
- $[s]_l \to s$,

- $[T]_{\mathsf{halt}} \to yes$,
- $[s]_{\mathsf{halt}} \to no$.

The first two rules are used in the countdown subtree (Figure 5.1c). From the very beginning of the computation, an l-membrane containing an object s gets dissolved in each computational step. Since the countdown subtree contains $7n + 2m + 5$ l-membranes, it takes $7n + 2m + 6$ steps for s to dissolve the h-membrane and all the l-membranes. On the other hand, it can be seen that if F is *true*, then exactly $7n + 2m + 5$ steps are necessary for a T-copy to reach the halt-membrane.

If F is *true* and thus a T-copy reaches the halt-membrane, then the system uses $[T]_{\mathsf{halt}} \to yes$ to introduce yes in the skin-membrane. In this case s arrives from the countdown subtree into the skin-membrane and the computation halts.

If F is *false* and thus none of the T-copies can reach the halt-membrane, then s arrives from the countdown subtree to the halt-membrane. Then s dissolves this membrane introducing no in the skin-membrane as the last step of the computation. With this, we have finished the description of $\Pi(n, m)$, and the correctness of the family $\mathbf{\Pi}$ has also been justified. To conclude the proof of Theorem 3, observe that

- the P systems in $\mathbf{\Pi}$ can be constructed by a logarithmic-space Turing machine,
- the input encoding cod can be constructed by a logarithmic-space Turing machine, and
- for each input formula F with n variables and m clauses, $\Pi(n, m)$ halts in linear time $O(n + m)$.

Hence, as logarithmic-space Turing machines are guaranteed to halt within a polynomial number of steps, our construction is indeed a polynomially uniform, polynomial-time family of polarizationless recognizer P systems, employing only in-communication, membrane dissolving and weak division rules, solving the PSPACE-complete QSAT problem.

5.5 Concluding remarks

In this chapter, we have investigated the computation power of polynomial-time polarizationless P systems with active membranes employing only in-communication, dissolution, and weak non-elementary membrane division rules. We have shown that these P systems can solve **PSPACE**-complete problems. We already pointed out in Chapter 1 that P systems with active membranes can solve exactly the problems in **PSPACE** in polynomial time [9, 50, 52]. Using our result and extending the proof idea of the **PSPACE** upper bound given in [52], we get that the polynomial-time P systems investigated in this chapter characterize **PSPACE**.

Chapter 6

Accepting conditions in P systems with active membranes

In this chapter, we introduce *elimination P systems* that are polarizationless P systems with active membranes extended with rules of the form $[ab \rightarrow \varepsilon]_h$, where a and b are objects, and ε denotes the empty word. Moreover, we present *extended acknowledger P systems* that are more general in terms of acceptance conditions than their recognizer counterpart. We will show that - along with some other important results - there are problems that recognizer elimination P systems cannot solve, but extended acknowledger elimination P systems can.

6.1 Introduction

Regarding Păun's conjecture and dissolution P systems, we already mentioned in Chapter 3 that one of the results presented in [19] was the fact that polarizationless P systems without dissolution rules can solve efficiently exactly the problems in $\mathbf{P}$. The $\mathbf{P}$ upper bound in this work was shown using the concept of dependency graphs of the P systems studied. For such a P system, a dependency graph is a directed graph where edges represent dependencies between objects on the left- and right-hand sides of the rules of the P system. Then one can verify whether yes appears in the output membrane by simply checking if yes can be reached from the relevant nodes in the dependency graph. However, this task is an instance of the $\mathbf{NL}$-complete REACHABILITY problem which we dealt with in Chapter 3. It is widely believed that $\mathbf{NL}$ is strictly included in $\mathbf{P}$. Nevertheless, this does not contradict the characterization of $\mathbf{P}$ appearing in [19], since it was noted in [34] that the $\mathbf{P}$ lower bound arises from the polynomial-time construction of the examined P systems. Therefore, it was suggested in [34] to consider stronger uniformity conditions in the case of P systems that solve problems in $\mathbf{P}$.

As we discussed in Chapter 3, in [35], $\mathbf{FAC^0}$-uniform families of recognizer po-

larizationless P systems without dissolution rules were examined, and it was shown that they solve only problems in $\mathbf{AC^0}$. To explore the potential effects of different acceptance conditions on the computational power of the P systems studied, [35] introduced a variant of recognizer P systems termed *acknowledger P systems*. These are P systems in which all computations halt and one or more copies of the distinguished object yes may or may not appear in the output membrane of the system. Moreover, these P systems have no rules applicable to yes. Recall that in recognizer P systems, exactly one no or yes must be produced in the output membrane, but not both, and only in the last step of the computation. In [35] it was found that acknowledger polarizationless P systems without dissolution rules decide exactly those languages that are $\mathbf{FAC^0}$ disjunctive truth table reducible to the unary languages in $\mathbf{NL}$. The fact that in [35] no similar result was obtained for these P systems using the more strict accepting conditions of recognizer P systems suggests that the accepting conditions have a fundamental impact on the computational power of the P systems under investigation. Using reasonably tight uniformity conditions has become standard in membrane computing when P systems solve problems in $\mathbf{P}$ (see, for example, our solution presented in Chapter 4, or the work [15] we recommend there and its references therein). However, the power of acknowledger P systems has received less attention when considering P systems solving problems beyond $\mathbf{P}$.

Păun's conjecture initiated another research line in membrane computing, which focuses on exploring efficiency frontiers in various classes of P systems. For example, in Chapter 3, we briefly discussed context-sensitivity used in [37], where bounded minimal cooperation rules are considered. These rules are of the form $[u \rightarrow v]_h$, where the length of v does not exceed that of u, and the lengths of both words range from one to two. As discussed above, polarizationless P systems with active membranes without dissolution and non-elementary membrane division cannot solve computationally hard problems efficiently. On the other hand, as stated in [37], these P systems running in polynomial time can solve the SAT problem (that is, the satisfiability problem of Boolean formulas) when endowed with bounded minimal cooperation rules.

In [13], another frontier of efficiency was identified for P systems with antimatter. These are polarizationless systems, where each object a has its corresponding antimatter $\overline{a}$, and they include annihilation rules of the form $[a\overline{a} \rightarrow \varepsilon]_h$. Such a rule is applicable when both a copy of a and $\overline{a}$ occur simultaneously in a membrane with label h, causing both particles to disappear from the membrane when applied. According to [13], P systems with antimatter without dissolution and non-elementary membrane division rules can efficiently solve the SAT problem if annihilation rules have priority over the other types of rules. However, without this priority, only problems in $\mathbf{P}$ can be efficiently solved by these P systems. It is easy to see that using reasonably tight uniformity conditions, the $\mathbf{P}$ upper bound given in [13] can be low-

ered to **NL** if annihilation rules have no priority.

In this chapter, we introduce a variant of P systems called elimination P systems and examine their computational power under different acceptance conditions. From a syntactic perspective, elimination P systems are polarizationless P systems with active membranes without non-elementary membrane division rules, extended with rules of the form $[ab \to \varepsilon]_h$, where a, b are objects and ε represents the empty word. Therefore, elimination rules are generalizations of the annihilation rules. The semantics of these rules is identical to that of non-prioritized annihilation rules: such a rule can be applied when both a and b are present at the same time in a membrane labeled with h, resulting in the elimination of both objects from the membrane upon application. On the other hand, elimination rules are weaker than bounded minimal communication rules with two objects on the left-hand side, since elimination rules cannot produce objects.

Our main results are as follows. We show that recognizer elimination P systems without dissolution rules can solve efficiently only problems in **NL**. We prove this by generalizing the notion of dependency graph and the corresponding proofs of [19] and [13]. In contrast to this, we show that polynomial-time recognizer elimination P systems with no communication rules can solve the **PP**-complete MAJORITY-SAT problem, where the task is to decide whether the majority of the truth assignments satisfy a Boolean formula or not. Moreover, we extend the notion of acknowledger P systems so that we allow them to include rules that can be applied to the output object yes, and show that extended acknowledger elimination P systems can solve MAJORITY-SAT in polynomial time even without dissolution rules. We also show that elimination P systems are naturally suitable for solving efficiently problem #SAT, the counting variant of SAT.

The remainder of the chapter is organized as follows. In the next section, we introduce the necessary notions and notations. Then, in Section 6.3, we present the above-mentioned results of the chapter. In Section 6.4, we make some concluding remarks.

6.2 Elimination P systems

In this chapter, we also consider the following type of rules, called *elimination* rules:

$$[ab \to \varepsilon]_h$$

for some $h \in H, a, b \in O$.

This rule is applicable only if the objects a and b are present in a membrane with the label h. The employment of this rule results in the disappearance of a and b from the h-membrane without any production. During a computation step,

these rules do not directly involve the membranes. Thus, they can be applied in the appropriate membranes, regardless of whether the membranes are used or not. To see the difference in the semantics of the evolution and elimination rules, we refer to Example 6.2.1. In what follows, we call those polarizationless P systems with active membranes that can also employ elimination rules *elimination P systems*.

Let $\Pi = (O, H, \mu, R)$ be an elimination P system and let C and C' be two configurations of Π. We write $C \Rightarrow C'$ to denote that Π can reach C' from C in one computation step (see Section 2.2). As usual, $\Rightarrow^*$ denotes the reflexive, transitive closure of $\Rightarrow$. Let $k \in \mathbb{N}$. Then $C \Rightarrow^k C'$ denotes that there are configurations $C_0, C_1, \ldots, C_k$ such that $C = C_0$, $C' = C_k$, and $C_i \Rightarrow C_{i+1}$ for every $i \in [0, k-1]$ (if $k = 0$, then the computation $C \Rightarrow^k C'$ contains no computation steps, only a configuration $C = C'$).

Example 6.2.1. Let $\Pi_{ex} = (\{a, b, c\}, \{h\}, \mu, \{[a \to c]_h, [ab \to \varepsilon]_h\})$ where $\mu = (\{x\}, \emptyset, L, \omega)$ with $L(x) = \mathsf{skin}$ and $\omega(x) = aab$. The two possible computations of Π_{ex} can be seen in Figure 6.1.

Figure 6.1: *The two possible computations of Π_{ex}. Circles denote membranes. In the first computation the rule $[a \to c]_h$ is applied only, while in the second computation both rules of Π_{ex} are employed.*

Let $\Pi = (O, H, \mu, R)$ be an elimination P system without dissolution rules, and let $a \in O, h \in H$. For a configuration C of Π and for an instance $\mathbf{a}$ of object a that occurs in an h-membrane $\mathbf{h}$ of C, $C^{(\mathbf{a},\mathbf{h})}$ denotes C with $\mathbf{a}$ being in $\mathbf{h}$. Let $C^{(\mathbf{a},\mathbf{m})}$ and $C'^{(\mathbf{b},\mathbf{n})}$ be two configurations of Π. By $C^{(\mathbf{a},\mathbf{m})} \prec C'^{(\mathbf{b},\mathbf{n})}$ we mean that $C \Rightarrow C'$ and

1. either $\mathbf{b}$ is the same in $\mathbf{n}$ as $\mathbf{a}$ in $\mathbf{m}$ (that is, no rule is applied on $\mathbf{a}$ during $C \Rightarrow C'$), or
2. $\mathbf{b}$ was introduced in $\mathbf{n}$ during $C \Rightarrow C'$ by a non-elimination rule applied on $\mathbf{a}$ in $\mathbf{m}$.

If Case 2 holds, then we say that $\mathbf{a}$ is *productive* in $C \Rightarrow C'$. By writing that $C^{(\mathbf{a},\mathbf{m})} \prec^* C'^{(\mathbf{b},\mathbf{n})}$ we mean that there are configurations $C_0^{(\mathbf{a}_0,\mathbf{h}_0)}, \ldots, C_l^{(\mathbf{a}_l,\mathbf{h}_l)}$, where $l \in \mathbb{N}$, such that $C_0^{(\mathbf{a}_0,\mathbf{h}_0)} = C^{(\mathbf{a},\mathbf{m})}$, $C_l^{(\mathbf{a}_l,\mathbf{h}_l)} = C'^{(\mathbf{b},\mathbf{n})}$, and for every $\kappa \in [0, l-1]$, $C_\kappa^{(\mathbf{a}_\kappa,\mathbf{h}_\kappa)} \prec C_{\kappa+1}^{(\mathbf{a}_{\kappa+1},\mathbf{h}_{\kappa+1})}$.

Acknowledger P systems were defined in [35] to generalize the accepting conditions of recognizer P systems commonly used in membrane computing to decide problems.

Definition 6.2.1. A P system Π is an *acknowledger* P system if it satisfies the following properties:

1. Π has a designated input membrane and a designated output membrane,
2. the alphabet of objects has a designated element yes,
3. Π has no rules that contain yes on the left-hand side, and
4. all computations of Π halt.

Let $\mathfrak{C}$ be a computation of Π. The output of Π related to $\mathfrak{C}$ is the multiset yes^t, where $t \in \mathbb{N}$ is the number of instances of yes in the output membrane of the halting configuration of $\mathfrak{C}$. Then $\mathfrak{C}$ is accepting if $t \neq 0$, otherwise $\mathfrak{C}$ is rejecting.

In this chapter, we consider a generalization of acknowledger P systems. We call a P system Π an *extended acknowledger* P system if Π satisfies only Conditions 1,2 and 4 in Definition 6.2.1.

Recognizer P systems (see Section 2.3) can be defined by restricting acknowledger P systems as follows.

Definition 6.2.2. A recognizer P system Π is an acknowledger P system that meets the following conditions:

1. in addition to the object yes, the alphabet of objects has a designated element no as well,
2. each computation of Π produces in the output membrane exactly one yes or no (but not both), and these objects are produced exactly in the last step of the computation.

Clearly, extended acknowledger P systems are a generalization of recognizer P systems.

Consider a recognizer P system Π and its halting computation $\mathfrak{C}$. Assume that the result of $\mathfrak{C}$ is yes^t for some $t \geq 0$. According to Definition 6.2.2, t can only be either one or zero. If $t = 1$, then $\mathfrak{C}$ represents an accepting computation. Conversely, if $t = 0$, then $\mathcal{C}$ is a rejecting computation, and consequently, the output membrane in $\mathfrak{C}$ must contain exactly one instance of the object no.

Next, we introduce the concept of an efficient solution for decision problems using **L**-uniform families of extended acknowledger P systems, similarly as in Definition 2.3.2.

Definition 6.2.3. Let R be a decision problem and $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ be an **L**-uniform family of extended acknowledger P systems. We say that $\mathbf{\Pi}$ *solves* R *in polynomial time* if the following conditions hold:

a) there is a deterministic Turing machine using logarithmic space computing the encoding function cod that transforms instances of R into multisets of objects,

b) there exists an integer $k \in \mathbb{N}$ such that for every instance x of R with size n, each computation $\mathfrak{C}$ of $\Pi(n)$ starting with $cod(x)$ in its input membrane halts in at most n^k steps, and

c) $\mathfrak{C}$ is accepting if and only if x is a positive instance of R.

P systems are naturally applicable to solve counting problems. The first approach to solving counting problems with membrane systems was in [7] where the $\#\mathbf{P}$-complete problem of counting the permanent of a binary matrix was solved by P systems with active membranes. In [7], the output of the presented P system was the unary representation of the solution. Counting P systems with binary output was introduced in [55]. In this chapter, we consider only the unary output case. For simplicity, counting P systems with unary output will be referred to hereafter as counting P systems.

Definition 6.2.4. A P system Π is a *counting* P system if it satisfies the following properties:

1. Π has a designated input membrane and a designated output membrane,
2. the alphabet of objects has a designated *output object*, and
3. all computations of Π halt.

Let $\mathfrak{C}$ be a computation of Π. The output of Π related to $\mathfrak{C}$ is the multiset σ^t, where σ is the output object and $t \in \mathbb{N}$ is the number of instances of σ in the output membrane of the halting configuration of $\mathfrak{C}$.

An $\mathbf{L}$-uniform family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ of counting P systems solves a counting problem R if conditions a) and b) of Definition 6.2.3 hold, moreover, for every instance x of R, the output of each computation of $\Pi(n)$ starting with $cod(x)$ in its input membrane is σ^t, $t \in \mathbb{N}$, if and only if t is the solution of x.

To simplify, hereafter, when we refer to solving a problem using P systems, we imply that the problem can be solved in polynomial time by an $\mathbf{L}$-uniform family of these P systems. For a given class $\mathcal{P}$ of P systems, $\mathbf{L} - \mathbf{PMC}_{\mathcal{P}}^{R}$ (respectively, $\mathbf{L} - \mathbf{PMC}_{\mathcal{P}}^{EA}$ and $\mathbf{L} - \mathbf{PMC}_{\mathcal{P}}^{C}$) denotes the family of problems that can be solved by recognizer (respectively, extended acknowledger and counting) P systems in $\mathcal{P}$.

6.3 Solving the variations of SAT with elimination P systems

The well-known $\mathbf{NP}$-complete problem SAT sounds as follows: given a propositional formula in CNF, decide whether it is satisfiable or not. In this chapter, we consider

two extensions of this problem: MAJORITY-SAT and #SAT. In the case of MAJORITY-SAT, the task is to determine for a given formula φ whether the majority of all truth assignments on Var_φ satisfy φ or not. MAJORITY-SAT is a **PP**-complete problem. Whereas MAJORITY-SAT is a decision problem, #SAT is a #**P**-complete counting problem, which sounds as follows: given a formula φ, count all the satisfying truth assignments of φ. Notice that the instances of SAT, MAJORITY-SAT and #SAT are the same.

Algorithm 4 presents a method to determine whether a truth assignment $\mathcal{I}$ satisfies φ. This method will be suitable for implementation by elimination P systems.

Algorithm 4 Checking $\mathcal{I} \models \varphi$

```
1: function checkSAT(φ, I)          ▷ φ is a formula with n variables and m clauses
2:   for i = 1...n do
3:     for j = 1...m do
4:       if I(x_i) = true and ¬x_i ∈ C_j then
5:         Delete(C_j,¬x_i)                          ▷ deleting ¬x_i from C_j
6:       end if
7:       if I(x_i) = false and x_i ∈ C_j then
8:         Delete(C_j,x_i)                           ▷ deleting x_i from C_j
9:       end if
10:    end for
11:  end for
12:  for j = 1...m do
13:    if |C_j| = 0 then
14:      return false
15:    end if
16:  end for
17:  return true
18: end function
```

In this section, we first demonstrate that recognizer elimination P systems without dissolution rules are capable of solving only problems in **NL**.

6.3.1 Recognizer elimination P systems with no dissolution rules

Denote $\mathcal{E}_{-d}$ the class of elimination P systems with no dissolution rules. Here, we show that $\mathbf{L} - \mathbf{PMC}^R_{\mathcal{E}_{-d}} \subseteq \mathbf{NL}$. The proof extends the similar ones found in [19] and [13], where the concept of dependency graphs was used to prove **P** upper bound on the computational power of the P systems under study. Hence, we first recall the definition of dependency graphs from [19].

Definition 6.3.1. Let $\Pi = (O, H, \mu, R)$ be an elimination P system without dissolution rules. The dependency graph $\mathcal{G}_\Pi$ of Π is a directed graph (V_Π, E_Π), $V_\Pi \subseteq O \times H$,

defined as follows. First, for every $h \in H$, let $p(h)$ be the label of the parent node of the membrane with label h in μ (if $h = \mathsf{skin}$, then let $p(h)$ be env denoting the environment). Then, let

$E_\Pi = E_{evo} \cup E_{in} \cup E_{out} \cup E_{div}$, where
$E_{evo} = \{((a,h),(b,h)) \mid \exists v \in O^* : b \in v \text{ and } [a \to v]_h \in R\}$,
$E_{out} = \{((a,h),(b,h')) \mid [a]_h \to [\ ]_h b \in R \text{ and } h' = p(h)\}$,
$E_{in} = \{((a,h),(b,h')) \mid a[\ ]_{h'} \to [b]_{h'} \in R \text{ and } h = p(h')\}$, and
$E_{div} = \{((a,h),(b,h)) \mid \exists c \in O : [a]_h \to [b]_h[c]_h \in R \text{ or } [a]_h \to [c]_h[b]_h \in R\}$, and let
$V_\Pi = \{(a,i) \in O \times H \mid \exists (b,j) \in O \times H : ((a,i),(b,j)) \in E_\Pi \text{ or } ((b,j),(a,i)) \in E_\Pi\}$.

For $(a,i),(b,j) \in V_\Pi$, $(a,i) \to (b,j)$ denotes that $((a,i),(b,j)) \in E_\Pi$, $\to^*$ denotes the reflexive, transitive closure of $\to$, and for a $k \in \mathbb{N}$, $(a,i) \to^k (b,j)$ denotes that there is a *walk* in $\mathcal{G}_\Pi$ from (a,i) to (b,j) containing k edges (if $k = 0$, then this walk does not contain edges, only node $(a,i) = (b,j)$).

Theorem 4. $\mathbf{L} - \mathbf{PMC}^R_{\mathcal{E}_{-d}} \subseteq \mathbf{NL}$.

Proof. Let $R \in \mathbf{L} - \mathbf{PMC}^R_{\mathcal{E}_{-d}}$. We show that $R \in \mathbf{NL}$. Consider an $\mathbf{L}$-uniform family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ of recognizer elimination P systems without dissolution rules that solves R in polynomial time. To see $R \in \mathbf{NL}$, it suffices to show the following. Let x be an input for R of size n and $cod(x)$ be an appropriate encoding of x. Then, the problem of deciding whether $\Pi(n)$ has an accepting computation on $cod(x)$ is in $\mathbf{NL}$.

Assume that $\Pi(n) = (O, H, \mu, R)$ with $\mu = (V, E, L, \omega)$. Let $\mathrm{in}, \mathrm{out} \in H$ be the labels of the input and output membranes of $\Pi(n)$, respectively. Moreover, let $\mathcal{G}_{\Pi(n)} = (V_{\Pi(n)}, E_{\Pi(n)})$ be the dependency graph of $\Pi(n)$. We claim that $\Pi(n)$ has an accepting computation on $cod(x)$ if and only if there is an object a in $cod(x) \cup \omega(\mathrm{in})$ such that $(a, \mathrm{in}) \to^* (yes, \mathrm{out})$. To see this, we show a more general statement, which we will refer to as (†): for every $(a,m),(b,n) \in O \times H$, $(a,m) \to^* (b,n)$ if and only if there are configurations $C^{(\mathbf{a},\mathbf{m})}, C'^{(\mathbf{b},\mathbf{n})}$ such that $C^{(\mathbf{a},\mathbf{m})} \prec^* C'^{(\mathbf{b},\mathbf{n})}$.

To see the forward direction of (†), assume that $(a,m) \to^* (b,n)$. Then, there is $k \in \mathbb{N}$ such that $(a,m) \to^k (b,n)$. We prove the forward direction of (†) by induction on k. Consider a configuration $C^{(\mathbf{a},\mathbf{m})}$. If $k = 0$, then $(a,m) = (b,n)$ and thus $C^{(\mathbf{a},\mathbf{m})} \prec^* C'^{(\mathbf{b},\mathbf{n})}$ holds trivially. Assume now that the forward direction of (†) holds for $k' = k-1$ when $k \geq 1$. Let $(c,h) \in V_{\Pi(n)}$ be such that $(a,m) \to^{k'} (c,h) \to (b,n)$. Then, by the induction hypothesis, $C^{(\mathbf{a},\mathbf{m})} \prec^* \hat{C}^{(\mathbf{c},\mathbf{h})}$ for a configuration $\hat{C}$. Since $(c,h) \to (b,n)$, there is a rule r of $\Pi(n)$ that can be triggered by $\mathbf{c}$ in $\mathbf{h}$, producing $\mathbf{b}$ in $\mathbf{n}$. Consider a configuration C' such that $\hat{C} \Rightarrow C'$ and r is applied on $\mathbf{c}$ in $\hat{C}$. Then $\hat{C}^{(\mathbf{c},\mathbf{h})} \prec C'^{(\mathbf{b},\mathbf{n})}$ and thus $C^{(\mathbf{a},\mathbf{m})} \prec^* C'^{(\mathbf{b},\mathbf{n})}$ also holds.

To see the backward direction of (†), let $C^{(\mathbf{a},\mathbf{m})}, C'^{(\mathbf{b},\mathbf{n})}$ be configurations of Π such

that $C^{(\mathbf{a},\mathbf{m})} \prec^* C'^{(\mathbf{b},\mathbf{n})}$. If $C^{(\mathbf{a},\mathbf{m})} = C'^{(\mathbf{b},\mathbf{n})}$, then $(a,m) = (b,n)$ and the statement is trivially valid. Otherwise, let $C_0^{(\mathbf{a_0},\mathbf{h_0})}, \ldots, C_l^{(\mathbf{a_l},\mathbf{h_l})}$, $l \geq 1$, be configurations such that $C_0^{(\mathbf{a}_0,\mathbf{h}_0)} = C^{(\mathbf{a},\mathbf{m})}$, $C_l^{(\mathbf{a}_l,\mathbf{h}_l)} = C'^{(\mathbf{b},\mathbf{n})}$, and for every $\kappa \in [0, l-1]$, $C_\kappa^{(\mathbf{a}_\kappa,\mathbf{h}_\kappa)} \prec C_{\kappa+1}^{(\mathbf{a}_{\kappa+1},\mathbf{h}_{\kappa+1})}$. Let $\{i_1, \ldots, i_k\}$, $i_1 < \ldots < i_k$, be the maximal subset of $[0, l-1]$ such that for every $m \in [k]$, $\mathbf{a}_{i_m}$ is productive in $C_{i_m} \Rightarrow C_{i_m+1}$. Using that $\Pi(n)$ has no dissolution rules and that for every $\kappa \in [0, i_1 - 1]$, $\mathbf{a}_\kappa$ is not productive in $C_\kappa \Rightarrow C_{\kappa+1}$, we get $(a_0, h_0) = (a_1, h_1) = \ldots = (a_{i_1}, h_{i_1})$. Notice that $\mathbf{a}_\kappa$ cannot be erased by an elimination rule as otherwise $C_\kappa^{(\mathbf{a}_\kappa,\mathbf{h}_\kappa)} \prec C_{\kappa+1}^{(\mathbf{a}_{\kappa+1},\mathbf{h}_{\kappa+1})}$ would not hold. Using a similar reasoning, we have $(a_{i_1+1}, h_{i_1+1}) = \ldots = (a_{i_2}, h_{i_2}), \ldots, (a_{i_k+1}, h_{i_k+1}) = \ldots = (a_l, h_l)$. Moreover, since for every $m \in [k]$, $\mathbf{a}_{i_m+1}$ is introduced in $\mathbf{h}_{i_m+1}$ during $C_{i_m} \Rightarrow C_{i_m+1}$ by a non-elimination rule applied to $\mathbf{a}_{i_m}$ in $\mathbf{h}_{i_m}$, we have $(a_{i_m}, h_{i_m}) \rightarrow (a_{i_m+1}, h_{i_m+1})$. Consequently, $(a_0, h_0) \rightarrow^* (a_l, h_l)$. That is, $(a, m) \rightarrow^* (b, n)$, which finishes the proof of the backward direction of (†).

To complete the proof, we only need to demonstrate that the problem of determining whether there exists an object $a \in cod(x) \cup \omega(\mathrm{in})$ such that $(a, \mathrm{in}) \rightarrow^* (yes, \mathrm{out})$ is in **NL**. This is substantiated by the following points:

1. $\mathcal{G}_{\Pi(n)}$ can be constructed using $O(\log n)$ space,
2. the size of $\mathcal{G}_{\Pi(n)}$ is $O(n^k)$ for some appropriate constant k,
3. the elements of $cod(x) \cup \omega(\mathrm{in})$ can be enumerated using $O(\log n)$ space, and
4. the reachability problem in $\mathcal{G}_{\Pi(n)}$ can be solved nondeterministically using logarithmic space in the size of $\mathcal{G}_{\Pi(n)}$, that is, in $O(\log n)$ space.

□

According to Theorem 4, and based on the commonly accepted assumption that $\mathbf{NL} \subsetneq \mathbf{NP}$, recognizer elimination P systems without using dissolution rules cannot solve hard problems efficiently. In the next part, we show that if we add the possibility of dissolving membranes to these P systems, then they can solve the **PP**-complete MAJORITY-SAT problem even without using communication rules.

6.3.2 The power of elimination P systems without communication rules

For the remainder of the chapter, let us fix a formula $\varphi = \mathcal{C}_1 \wedge \ldots \wedge \mathcal{C}_m$ with $n \geq 2$ variables and $m \geq 1$ clauses. First, we define an alphabet to encode φ:

$$\Sigma(n, m) = \{v_{i,j}, \overline{v}_{i,j} \mid i \in [n], j \in [m]\}.$$

Then, the encoding of φ, denoted by $cod(\varphi)$, is the following subset of $\Sigma(n, m)$:

$$cod(\varphi) = \{v_{i,j} \mid x_i \in \mathcal{C}_j\} \cup \{\overline{v}_{i,j} \mid \neg x_i \in \mathcal{C}_j\}.$$

It is not hard to see that there exists a logarithmic-space deterministic Turing machine that can compute $cod(\varphi)$ when it is started with an appropriate representation of φ on the input tape.

Next, we define a set containing objects we will frequently use in our P systems:

$$O = \Sigma(n,m) \cup \{a_i, t_i, f_i, \hat{t}_i, \hat{f}_i \mid i \in [n]\} \cup \{c_j, \hat{c}_i \mid j \in [m], i \in [n+1]\} \cup \{e_i \mid i \in [n+3]\}.$$

Clearly, there is a natural relationship between the solutions for the same instance φ of #SAT and MAJORITY-SAT. In the case of #SAT, it is necessary to count the truth assignments that satisfy φ, whereas for MAJORITY-SAT, one must determine if more than half of the potential truth assignments satisfy φ. Specifically, if a P system can determine the number t of truth assignments satisfying φ, then, to determine whether the majority of the truth assignments satisfy φ, the P system needs to compare the values t and 2^{n-1}, where $n = |Var_\varphi|$. Therefore, we first solve #SAT with a family of counting elimination P systems, and based on this, we solve MAJORITY-SAT with a family of recognizer elimination P systems. Denote $\mathcal{E}_{-c}$ the class of elimination P systems with no communication rules.

Theorem 5. $\#\mathbf{P} \subseteq \mathbf{L} - \mathbf{PMC}^C_{\mathcal{E}_{-c}}$.

Proof. We prove Theorem 5 by showing that #SAT can be solved by an **L**-uniform family $\mathbf{\Pi}^C_{-c} = \{\Pi^C_{-c}(n,m) \mid n \geq 2, m \geq 1\}$ of counting elimination P systems without communication rules. We define $\Pi^C_{-c}(n,m) = (O^C_{-c}, H, \mu, R^C_{-c})$ as follows:

- $O^C_{-c} = O \cup \{e, d, \hat{d}\}$,
- the output object is e,
- $H = \{\mathsf{skin}, h\}$,
- the initial membrane configuration μ is given in Figure 6.2, where the labels of the input and output membranes are h and skin, respectively, and
- the rules of R^C_{-c} are given in Table 6.1.

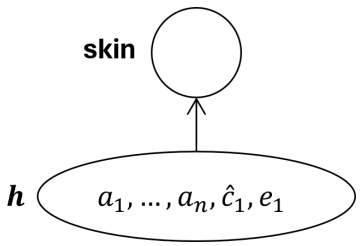


Figure 6.2: *The initial membrane configuration of* $\Pi^C_{-c}(n,m)$.

1. $[a_i]_h \to [\hat{t}_i]_h[\hat{f}_i]_h$,
2. $[\hat{t}_i \to t_i^m]_h$,
3. $[\hat{f}_i \to f_i^m]_h$,

$\Big\}\ i \in [n]$

4. $[\hat{c}_j \to \hat{c}_{j+1}]_h, \quad (j \in [n+1])$
5. $[\hat{c}_{n+2} \to c_1 \dots c_m]_h$,

(a) *Rules for the generation stage*

6. $[t_i\overline{v}_{i,j} \to \varepsilon]_h$,
7. $[f_i v_{i,j} \to \varepsilon]_h$,
8. $[c_j\overline{v}_{i,j} \to \varepsilon]_h$,
9. $[c_j v_{i,j} \to \varepsilon]_h$,

$\Big\}\ i \in [n], j \in [m]$

10. $[e_k \to e_{k+1}]_h, \quad (k \in [n+2])$
11. $[e_{n+3} \to e\hat{d}]$,
12. $[ec_j \to \varepsilon]_h, \quad (j \in [m])$,
13. $[\hat{d} \to d]_h$,
14. $[d]_h \to d$.

(b) *Rules for the evaluation stage*

Table 6.1: *Rules of* $\Pi^C_{-c}(n,m)$

An overview of the computation.

Assume that $\Pi^C_{-c}(n,m)$ is started with the multiset $cod(\varphi)$ in the h-membrane. We describe the computation of $\Pi^C_{-c}(n,m)$ and along with it, we also show its correctness (see also Example 6.3.1 for a demonstration of the working of $\Pi^C_{-c}(2,2)$). We can split the computation of $\Pi^C_{-c}(n,m)$ into two phases that are described in the following.

Generation phase. In the first n steps, $\Pi^C_{-c}(n,m)$ generates 2^n h-membranes by applying the rules $[a_i]_h \to [\hat{t}_i]_h[\hat{f}_i]_h$ for every $i \in [n]$ (refer to the first rule in Section (a) of Table 6.1). When a rule $[a_i]_h \to [\hat{t}_i]_h[\hat{f}_i]_h$ is applied, objects $\hat{t}_i$ and $\hat{f}_i$ are introduced in the two new membranes, respectively. Then, during the next step, $\hat{t}_i$ (resp. $\hat{f}_i$) evolves to m copies of t_i (resp. f_i) by applying the second (resp. the third) rule in Section (a). Therefore, at the end of the $n+1$ steps, each h-membrane holds m copies of either t_i or f_i (exclusively) for each $i \in [n]$. This arrangement represents a unique truth assignment to the variables in φ, where *true* or *false* is assigned to x_i

accordingly. We call these objects *evaluation objects* in what follows.

In parallel to the generation of evaluation objects, the rules $[\hat{c_j} \to \hat{c}_{j+1}]_h$, $j \in [n+1]$, are used to count the steps of the P system. In the $(n+2)th$ step, the multiset $c_1 \dots c_m$ that denotes the clauses of φ is introduced in each h-membrane. We refer to these objects as *clause objects* hereafter.

Evaluation phase. In this phase, $\Pi^C_{-c}(n,m)$ initially eliminates objects from $cod(\varphi)$ that represent *false* literals according to the corresponding truth assignment for φ, by applying the first two rules given in Section (b) of Table 6.1 (cf. also Lines 2–11 of Algorithm 4). This process runs in parallel with the generation phase. The fact that the evaluation objects are introduced in m copies in that phase guarantees that any objects of the forms $v_{i,j}$ and $\overline{v}_{i,j}$, representing *false* literals, are removed. This elimination process is completed in $n+2$ steps.

Recall that clause objects are introduced during the $(n+2)$th step. Therefore, after $n+2$ steps, the objects of the forms $v_{i,j}$ and $\overline{v}_{i,j}$ representing *true* literals and the clause objects are present in each h-membrane. Then, in the $(n+3)$th step, $\Pi^C_{-c}(n,m)$ eliminates all clause objects that represent *true* clauses of φ by applying the rules $[c_j\overline{v}_{i,j} \to \varepsilon]_h$ and $[c_j v_{i,j} \to \varepsilon]_h$ of Section (b) in Table 6.1.

Consequently, after $n+3$ steps, an h-membrane M holds a clause object if and only if the truth assignment $\mathcal{I}$ represented by M does not satisfy φ. In contrast, our goal is for M to contain exactly one copy of a specific object (which will be object e) if and only if $\mathcal{I}$ satisfies φ. To this end, the rules $[e_k \to e_{k+1}]_h$, $k \in [n+2]$, and $[e_{n+3} \to e\hat{d}]_h$ are used to count the steps of the P system until the e-copies and $\hat{d}$-copies are introduced in each h-membrane in the $(n+3)$th step. Then, in the next step, in each h-membrane, the $\hat{d}$-copy evolves to a d-copy, and the e-copy is removed by a rule $[ec_j \to \varepsilon]_h$ on the condition that a clause object still occurs in the same h-membrane (refer to the 12th and 13th rules in Table 6.1).

That is, after $n+4$ steps, an h-membrane M contains exactly one e-copy if and only if M represents a satisfying truth assignment of φ. Finally, in the last step, the h-membranes are dissolved by the last rule in Table 6.1. Consequently, after $n+5$ steps, the skin-membrane contains the multiset e^t, where t is the number of truth assignments that satisfy φ.

Regarding the **L**-uniformity of $\mathbf{\Pi}^C_{-c}$, we observe the following points. Initially, the membrane structure μ consists of two membranes. The skin-membrane of μ contains no objects, whereas the h-membrane contains $O(n)$ objects. Consequently, μ can be constructed using $O(\log n)$ space. Furthermore, the sizes of O^C_{-c} and R^C_{-c} are $O(n+m)$ and $O(nm)$, respectively. Hence, the elements of both O^C_{-c} and R^C_{-c} can be generated using $O(\log(nm))$ space. Thus, $\Pi^C_{-c}(n,m)$ can be created using $O(\log(nm))$, demonstrating its **L**-uniformity. With this, we have finished the proof of Theorem 5. □

Example 6.3.1. Let $\varphi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$. Then, $cod(\varphi) = \{v_{1,1}, v_{2,1}, v_{1,2}, \overline{v}_{2,2}\}$ and the computation of $\Pi^C_{-c}(2,2)$ with input $cod(\varphi)$ can be seen in Figure 6.3.

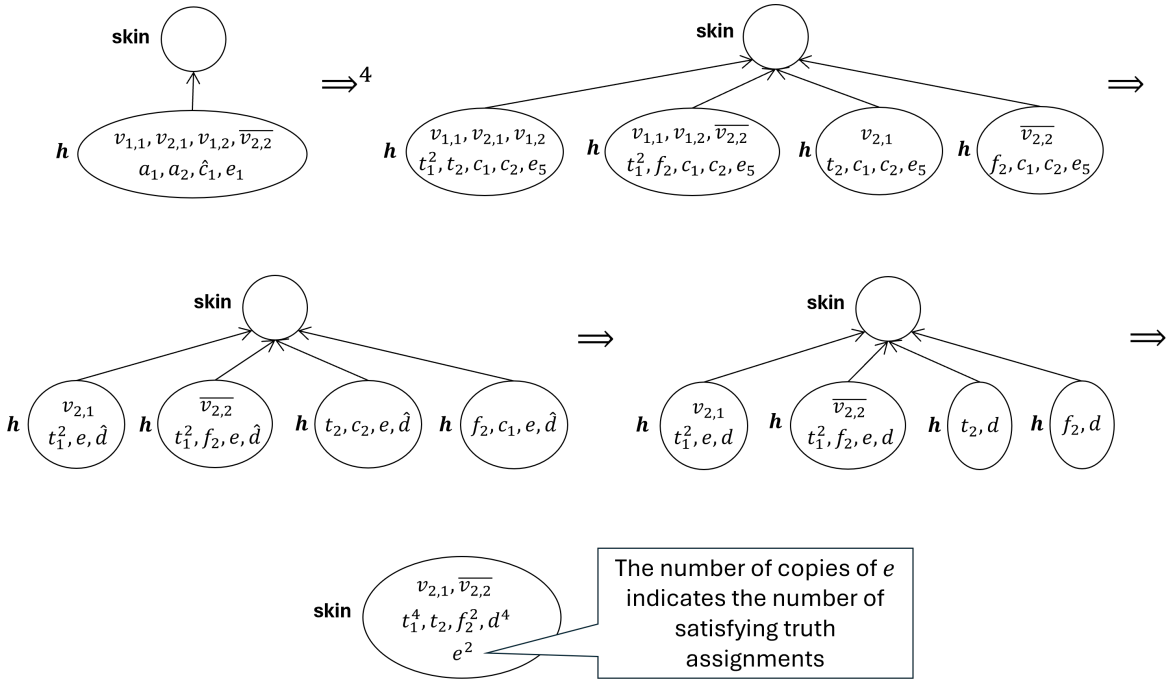


Figure 6.3: *A computation of* $\Pi^C_{-c}(2,2)$ *with input multiset encoding the formula* $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$.

Next, we demonstrate that $\Pi^C_{-c}(n,m)$ can be generalized to solve **PP**-complete problems.

Theorem 6. $\mathbf{PP} \subseteq \mathbf{L} - \mathbf{PMC}^R_{\mathcal{E}_{-c}}$.

Proof. We prove Theorem 6 by showing that MAJORITY-SAT can be solved by an **L**-uniform family $\mathbf{\Pi}^R_{-c} = \{\Pi^R_{-c}(n,m) \mid n \geq 2, m \geq 1\}$ of recognizer elimination P systems without communication rules. We define $\Pi^R_{-c}(n,m) = (O^R_{-c}, H, \mu, R^R_{-c})$ as follows:

- $O^R_{-c} = O \cup \{e, \bar{e}, d, \hat{d}, g_1, \ldots, g_6, g, yes, no\} \cup \{\bar{e}_k \mid k \in [n]\} \cup \{no_k \mid k \in [n+8]\}$,
- $H = \{\mathsf{skin}, h, h_1, h_2\}$,
- the initial membrane configuration μ can be seen in Figure 6.4, where the labels of the input and output membranes are h and skin, respectively, and
- the rules of R^R_{-c} are all the rules of $\Pi^C_{-c}(n,m)$ and the following ones:

  15. $[\bar{e}_k \to \bar{e}_{k+1}\bar{e}_{k+1}]_{h_1}, \quad (k \in [n-1])$,

16. $[\bar{e}_n \to \bar{e}g_1]_{h_1}$,
17. $[e\bar{e} \to \varepsilon]_{h_1}$,
18. $[g_k \to g_{k+1}]_{h_1}, \quad (k \in [5])$,
19. $[g_6]_{h_1} \to g$,
20. $[no_k \to no_{k+1}]_{h_2}, \quad (k \in [n+7])$,
21. $[redno_{n+8}]_{h_2} \to no$,
22. $[e]_{h_2} \to yes$.

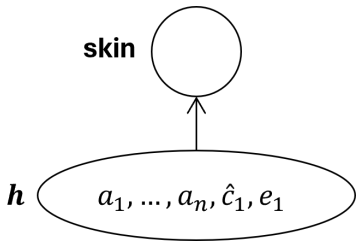


Figure 6.4: *The initial membrane configuration of* $\Pi^R_{-c}(n,m)$.

An overview of the computation.

Assume again that $\Pi^R_{-c}(n,m)$ is started with the multiset $cod(\varphi)$ in the h-membrane. Given that the first 14 rules of $\Pi^R_{-c}(n,m)$ are identical to those of $\Pi^C_{-c}(n,m)$, using the discussion in the proof of Theorem 5 on the behavior of $\Pi^C_{-c}(n,m)$, we can infer the following: After $n+5$ steps, the h_1-membrane of $\Pi^R_{-c}(n,m)$ will contain the multiset e^t, where t represents the number of truth assignments that satisfy φ (see also Figure 6.5 for an example computation of $\Pi^R_{-c}(2,2)$).

On the other hand, using rules 15 and 16, $\Pi^R_{-c}(n,m)$ produces 2^{n-1} $\bar{e}$-copies and the same number of g_1-copies in the h_1-membrane. This process takes n steps. During the next five steps, g_1 evolves to g_6 due to rule 18. Recall that the multiset e^t appears in the h_1-membrane in the $(n+5)$th step. Thus, during the next step, the following happens. On the one hand, rule $[e\bar{e} \to \varepsilon]_{h_1}$ (rule 17) is applied to $\min\{t, 2^{n-1}\}$ e-copies. On the other hand, an object g_6 dissolves the h_1-membrane. Therefore, after $n+6$ steps, the h_2-membrane contains an e-copy if and only if more than half of the possible truth assignments satisfy φ, that is, φ is a positive instance of MAJORITY-SAT. If so, then one e-copy dissolves the h_2-membrane (rule 22) becoming object yes in the skin-membrane.

If φ is a negative instance of MAJORITY-SAT, then the following happens. Since in this case the h_2-membrane contains no object e after $n+6$ steps, $\Pi^R_{-c}(n,m)$ still has the h_2-membrane after $n+7$ steps. Thus, using rules $[no_k \to no_{k+1}]_{h_2}$ ($k \in [n+7]$, see rule 20), object no_1 evolves to object no_{n+8} in the h_2-membrane. Then, using

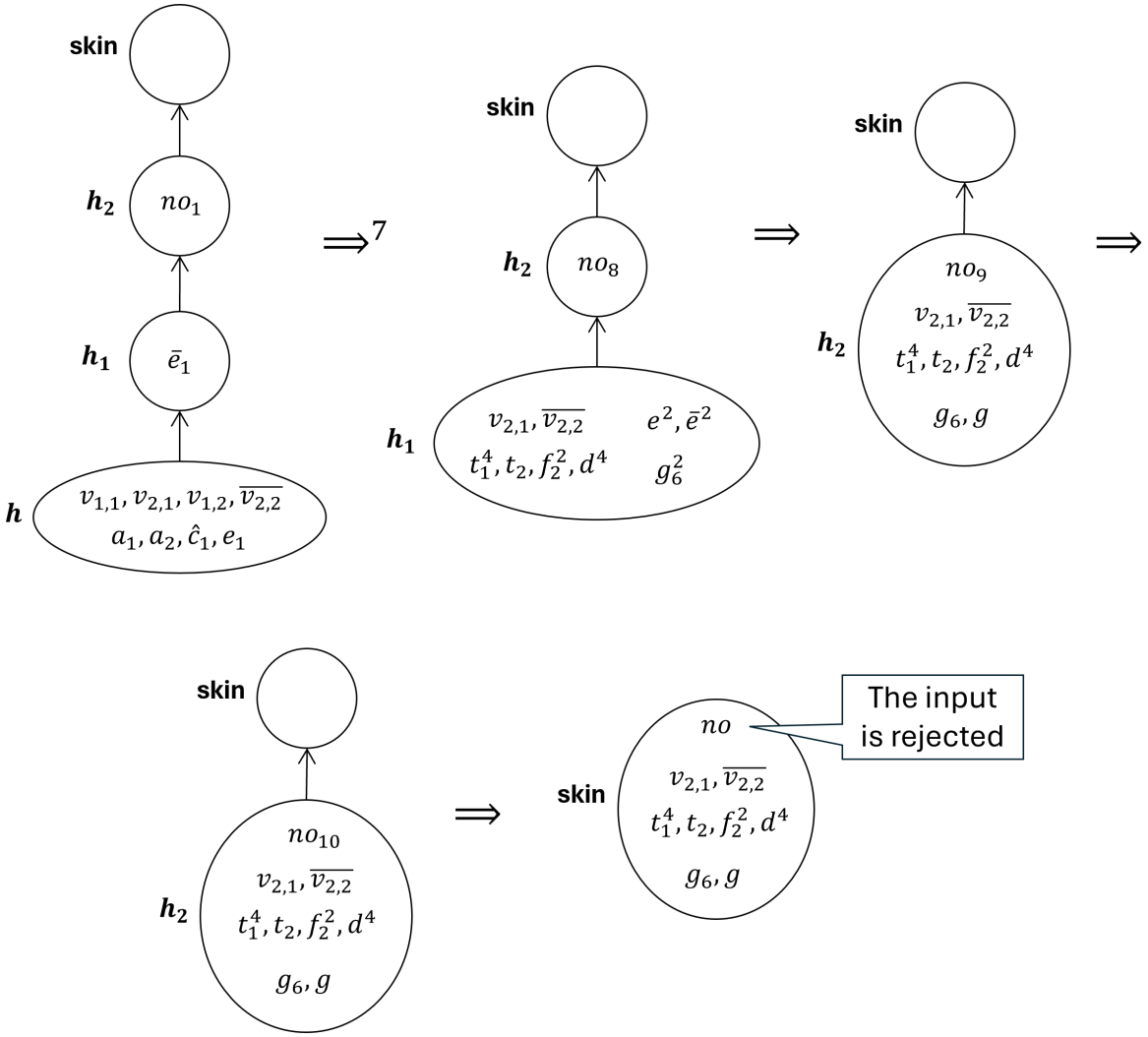


Figure 6.5: *A computation of* $\Pi^R_{-c}(2,2)$ *with input multiset encoding the formula* $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$.

the rule $[no_{n+8}] \to no$ (rule 21), $\Pi^R_{-c}(n,m)$ dissolves this membrane introducing an object no in the skin-membrane. The **L**-uniformity of $\mathbf{\Pi}^R_{-c}$ can be seen similarly as the **L**-uniformity of $\mathbf{\Pi}^C_{-c}$ was proved at the end of the proof of Theorem 5. With this, we have finished the proof of Theorem 6. □

6.3.3 The power of elimination P systems without dissolution rules

In this part, we first demonstrate that elimination P systems without dissolution rules are capable of counting the satisfying truth assignments of a formula. Furthermore, we show that by employing the acceptance condition described in Definition 6.2.1,

these P systems can also solve the MAJORITY-SAT problem.

Theorem 7. $\#\mathbf{P} \subseteq \mathbf{L} - \mathbf{PMC}^{C}_{\mathcal{E}_{-d}}$

Proof. To prove Theorem 7, we show that $\#\text{SAT} \in \mathbf{L} - \mathbf{PMC}^{C}_{\mathcal{E}_{-d}}$. For this, we construct an $\mathbf{L}$-uniform family $\mathbf{\Pi}^{C}_{-d} = \{\Pi^{C}_{-d}(n,m) \mid n \geq 2, m \geq 1\}$ of elimination P systems with no dissolution rules which solves $\#\text{SAT}$. This construction relies on $\mathbf{\Pi}^{C}_{-c}$ given in the proof of Theorem 5. In particular, the initial twelve rules of the corresponding P systems in these families are identical. Let $\Pi^{C}_{-d}(n,m) = (O^{C}_{-d}, H, \mu, R^{C}_{-d})$ where:

- $O^{C}_{-d} = O \cup \{s_k \mid k \in [n+1]\} \cup \{e, d, \hat{d}, \sigma\}$,
- the output object is σ,
- $H = \{\mathsf{skin}, h\}$,
- the initial membrane configuration μ is defined similarly as in Figure 6.2, except that now the skin-membrane contains an s_1-copy,
- the labels of the input and output membranes are h and skin, respectively, and
- the set of rules R^{C}_{-d} is the rules 1-12 given in Table 6.1 and the following ones:

  13. $[\hat{d} \to dd]_h$,
  14. $[ed \to \varepsilon]_h$,
  15. $[d]_h \to [\,]_h d$,
  16. $[s_k \to s_{k+1}s_{k+1}]_{\mathsf{skin}}, \quad (k \in [n])$
  17. $[s_{n+1} \to \sigma\sigma]_{\mathsf{skin}}$,
  18. $[\sigma d \to \varepsilon]_{\mathsf{skin}}$.

An overview of the computation.

Assume again that $\Pi^{C}_{-d}(n,m)$ is started with the multiset $cod(\varphi)$ in the h-membrane. Since rules 1-12 of $\Pi^{C}_{-d}(n,m)$ are the same as those of $\Pi^{C}_{-c}(n,m)$, using the discussion given in the proof of Theorem 5 on the behavior of $\Pi^{C}_{-c}(n,m)$, we can conclude the following. Consider an h-membrane M after $n+3$ steps of $\Pi^{C}_{-d}(n,m)$. Then

- M contains a clause object if and only if for the truth assignment $\mathcal{I}$ represented by M, $\mathcal{I}(\varphi) = false$ and
- M contains an object e along with an object $\hat{d}$ (see also Figure 6.6 for an example computation of $\Pi^{C}_{-d}(2,2)$).

Then, the computation is driven by rules 13-18 as follows. In the $(n+4)$th step, the object $\hat{d}$ evolves to dd according to rule 13. One of these d-copies exits from

each h-membrane during the next step according to rule 15, and the other d-copy is eliminated by the rule $[ed \to \varepsilon]_h$ (rule 14) in those h-membranes that contain an object e and thus represent a satisfying truth assignment. Therefore, after $n+5$ steps, the skin-membrane contains 2^n d-copies, and each h-membrane M contains an object d if and only if M represents an unsatisfying truth assignment. During the next step, the h-membranes containing d emit this object to the skin-membrane by using rule 15 again.

Meanwhile, in the initial n steps of the computation, $\Pi^C_{-d}(n,m)$ generates 2^n copies of s_{n+1} using rule 16. Then, in the next step, 2^{n+1} copies of σ are produced by the application of rule 17. Recall that in the $(n+5)$th step, 2^n d-copies appear in the skin-membrane. Consequently, during the $(n+6)$th step, the following happens in this membrane. On the one hand, these 2^n d-copies are eliminated by the rule $[\sigma d \to \varepsilon]_{\mathsf{skin}}$ (rule 18). On the other hand, $2^n - t$ d-copies appear in the skin-membrane, where t represents the number of truth assignments satisfying φ. Hence, after the $(n+6)$th step, the skin-membrane contains 2^n copies of σ and $2^n - t$ d-copies. Subsequently, in the final step, using rule 18, $\Pi^C_{-d}(n,m)$ removes all d-copies together with $2^n - t$ copies of σ. As a result, the remaining t copies of σ correctly indicate the number of truth assignments satisfying φ. The $\mathbf{L}$-uniformity of $\mathbf{\Pi}^C_{-d}$ can be seen similarly as the $\mathbf{L}$-uniformity of $\mathbf{\Pi}^C_{-c}$ was proved at the end of the proof of Theorem 5. With this, we concluded the proof of Theorem 7. □

Extending the P systems in $\mathbf{\Pi}^C_{-d}$ with the appropriate rules, we can prove the following result.

Theorem 8. $\mathbf{PP} \subseteq \mathbf{L} - \mathbf{PMC}^{EA}_{\mathcal{E}_{-d}}$.

Proof. To prove this statement, we show that MAJORITY-SAT $\in \mathbf{L} - \mathbf{PMC}^{EA}_{\mathcal{E}_{-d}}$. To this end, we define a family $\mathbf{\Pi}^{EA}_{-d} = \{\Pi^{EA}_{-d}(n,m) \mid n \geq 2, m \geq 1\}$ of extended acknowledger elimination P systems capable of solving MAJORITY-SAT by slightly modifying $\mathbf{\Pi}^C_{-d}$ defined in the previous section. Let $\Pi^{EA}_{-d}(n,m) = (O^{EA}_{-d}, H, \mu, R^{EA}_{-d})$ where:

- $O^{EA}_{-d} = (O^C_{-d} - \{\sigma\}) \cup \{z_k \mid k \in [n-1]\} \cup \{yes, \overline{yes}_1, \ldots, \overline{yes}_7, \overline{yes}\}$,
- $H = \{\mathsf{skin}, h\}$,
- the initial membrane configuration μ is defined similarly as in Figure 6.2, except that now the skin-membrane contains one copy each of s_1 and z_1,
- the labels of the input and output membranes are h and skin, respectively, and
- the set of rules R^{EA}_{-d} consists of the first 16 rules of R^C_{-d} defined in the proof of Theorem 7 and the following ones:

  17. $[s_{n+1} \to yes\, yes]_{\mathsf{skin}}$,
  18. $[yes\, d \to \varepsilon]_{\mathsf{skin}}$,

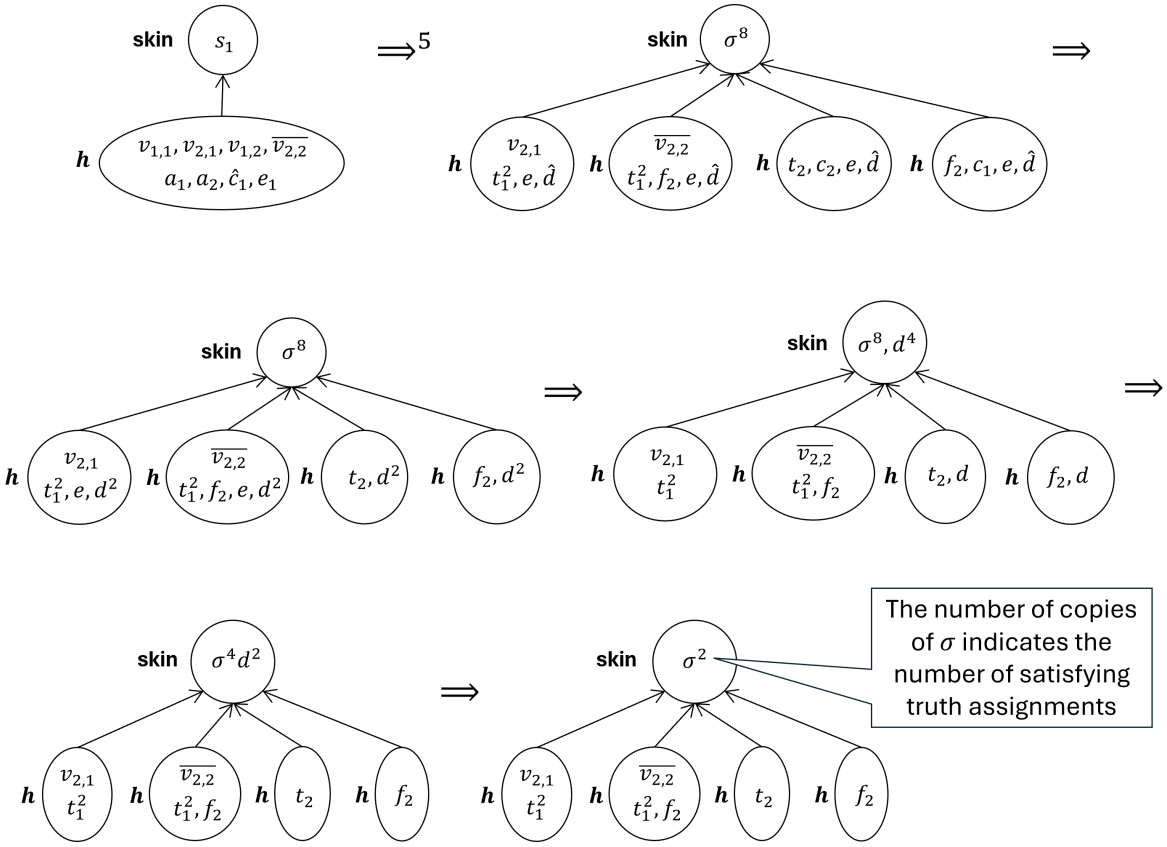


Figure 6.6: *A computation of* $\Pi^C_{-d}(2,2)$ *with input multiset encoding the formula* $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$.

19. $[z_k \to z_{k+1} z_{k+1}]_{\text{skin}}, \quad (k \in [n-1])$
20. $[z_n \to \overline{yes}_1]_{\text{skin}},$
21. $[\overline{yes}_k \to \overline{yes}_{k+1}]_{\text{skin}}, \quad (k \in [6])$
22. $[\overline{yes}_7 \to \overline{yes}]_{\text{skin}},$
23. $[yes\, \overline{yes} \to \varepsilon]_{\text{skin}}.$

An overview of the computation.

Assume again that $\Pi^{EA}_{-d}(n,m)$ is started with the multiset $cod(\varphi)$ in the h-membrane. Recall that the first 16 rules for both $\Pi^{EA}_{-d}(n,m)$ and $\Pi^C_{-d}(n,m)$ are identical by definition, and notice that the 17th and 18th rules differ in that $\Pi^C_{-d}(n,m)$ evolves s_{n+1} to σ and subsequently eliminates σ with the aid of object d, whereas $\Pi^{EA}_{-d}(n,m)$ evolves s_{n+1} to yes and then eliminates yes using d. Therefore, using the discussion given in the proof of Theorem 7 about the behavior of $\Pi^C_{-d}(n,m)$, we can conclude that after $n+7$ steps of $\Pi^{EA}_{-d}(n,m)$, the number of copies of object yes in the skin-membrane is equal to the number t of satisfying truth assignments of φ (see also Figure 6.7 for an

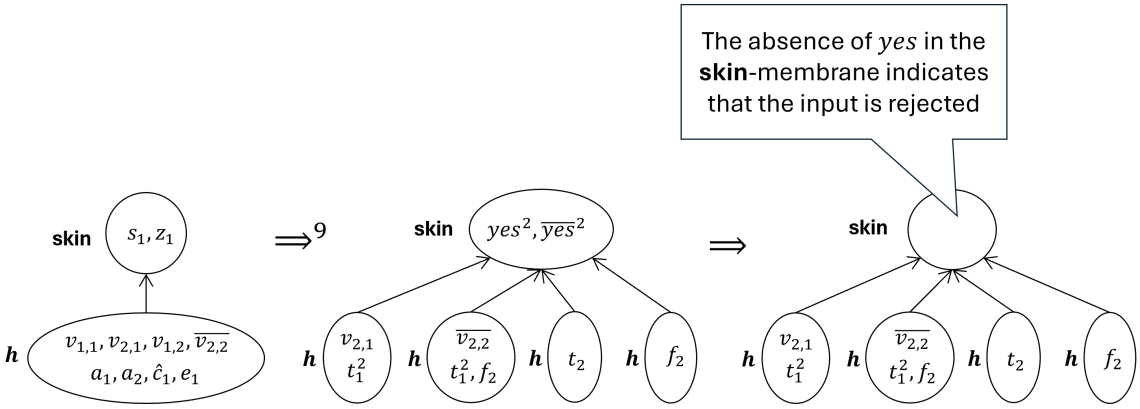


Figure 6.7: *A computation of* $\Pi_{-d}^{EA}(2,2)$ *with input multiset encoding the formula* $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$.

example computation of $\Pi_{-d}^{EA}(2,2)$).

On the other hand, using the rules $[z_k \to z_{k+1}z_{k+1}]_{\text{skin}}$ and $[z_n \to \overline{yes}_1]_{\text{skin}}$ (rules 19 and 20), $\Pi_{-d}^{EA}(n,m)$ produces 2^{n-1} copies of $\overline{yes}_1$ in the skin-membrane during the first n steps. Then, each copy of $\overline{yes}_1$ evolves to $\overline{yes}$ in seven steps by the rules $[\overline{yes}_k \to \overline{yes}_{k+1}]_{\text{skin}}$ and $[\overline{yes}_7 \to \overline{yes}]_{\text{skin}}$ (rules 21 and 22). In the $(n+8)$th step, $\Pi_{-d}^{EA}(n,m)$ applies the rule $[yes\,\overline{yes} \to \varepsilon]_{\text{skin}}$ (rule 23) to remove 2^{n-1} copies of yes. That is, after $n+8$ steps, the skin-membrane contains at least one yes if and only if $t > 2^{n-1}$, that is, φ is true in more than half of the possible truth assignments. The $\mathbf{L}$-uniformity of $\mathbf{\Pi}_{-d}^{EA}$ can be demonstrated in a similar manner to the $\mathbf{L}$-uniformity of $\mathbf{\Pi}_{-c}^{C}$, as was shown at the end of the proof of Theorem 5. □

6.4 Concluding remarks

In this chapter, we investigated the computational power of elimination P systems, which are a kind of polarizationless P system with active membranes. We showed that, without dissolution rules, recognizer elimination P systems are capable of solving efficiently only problems in $\mathbf{NL}$, while the inclusion of dissolution rules enables them to solve the $\mathbf{PP}$-complete MAJORITY-SAT problem in polynomial time even without communication rules.

We also demonstrated that polynomial-time elimination P systems are capable of solving MAJORITY-SAT without dissolution rules, provided that the acceptance conditions are defined by those of extended acknowledger P systems.

This chapter leaves several interesting questions unanswered. For example, are acknowledger P systems without dissolution rules capable of solving MAJORITY-SAT efficiently? Recall that in acknowledger P systems, no rule can be applied on the output symbol yes. Furthermore, are the $\mathbf{PP}$-lower bounds established in this chapter

tight or not? Addressing these questions is part of our future research plan.

Bibliography

Journal publications of the author

[1] Gazdag, Zs., **Hajagos, K.**, Iván, Sz.: On the power of P systems with active membranes using weak non-elementary membrane division. *Journal of Membrane Computing*, **3**, 258–269 (2021)

[2] Gazdag, Zs., **Hajagos, K.**: On the power of membrane dissolution in polarizationless P systems with active membranes. *Natural Computing*, 22(1), 1-10 (2022).

[3] Gazdag, Zs., **Hajagos, K.**: A characterisation of P by DLOGTIME-uniform families of polarizationless P systems using only dissolution rules. *Theoretical Computer Science*, 965(3), (2023)

[4] Gazdag, Zs., **Hajagos, K.**: On accepting conditions in P systems with active membranes. *Journal of Membrane Computing*- Accepted, soon to be published - (2025)

Further related publications of the author

[5] Gazdag, Zs., **Hajagos, K.**, Iván, Sz.: On the number of useful objects in P systems with active membranes. https://arxiv.org/abs/2008.04993 (2020)

[6] Gazdag, Zs., **Hajagos, K.**: A characterisation of $\mathbf{P}$ by polarizationless P systems using only membrane dissolution rules. Presented at the 23rd Conference on Membrane Computing (CMC 2022), 6-9 September 2022, Trieste, Italy

Other references

[7] Alhazov, A., Burtseva, L., Cojocaru, S., Rogozhin, Y.: Solving PP-Complete and #P-Complete Problems by P Systems with Active Membranes. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds) Membrane Computing. WMC 2008. Lecture Notes in Computer Science, vol 5391. Springer, Berlin (2009)

[8] Alhazov, A., Leporati, A., Manzoni, L., Mauri, G., Zandron, C.: Alternative space definitions for P systems with active membranes. Journal of Membrane Computing 3: 87-96 (2021)

[9] Alhazov, A., Martín-Vide, C., Pan, L.: Solving a PSPACE-complete problem by P systems with restricted active membranes. Fundamenta Informaticae **58**, 67–77 (2003)

[10] Alhazov, A., Pan, L., Păun, Gh.: Trading polarizations for labels in P systems with active membranes. Acta Informatica **41**(2-3), 111–144 (2004)

[11] Alhazov, A., Pérez-Jiménez, M.J.: Uniform solution of QSAT using polarizationless active membranes. International Conference on Machines, Computations and Universality, 122-133 (2007)

[12] Csuhaj-Varjú, E., Vaszil, Gy.: P Automata or Purely Communicating Accepting P Systems. Lecture Notes in Computer Science, 2597, 219–233 (2003)

[13] Díaz-Pernil, D., Alhazov, A., Freund, R, Gutiérrez-Naranjo, M. A., Leporati, A.: Recognizer P Systems with Antimatter. Romanian Journal of Information Science and Technology 18(3), 201–217 (2015)

[14] Gazdag, Z., Gutiérrez-Naranjo, M.A.: Solving the ST-connectivity problem with pure membrane computing techniques. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) Membrane Computing: 15th International Conference, LNCS vol. 8961, 215–228 (2014)

[15] Gazdag, Z., Kolonits, G.: Remarks on the computational power of some restricted variants of P systems with active membranes. In: Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing, 17th International Conference, LNCS vol. 10105, 209–232 (2017)

[16] Gazdag, Zs., Kolonits, G.: A new method to simulate restricted variants of polarizationless P systems with active membranes. Journal of Membrane Computing 1(4): 251–261 (2019)

[17] Gazdag, Z., Kolonits, G., Gutiérrez-Naranjo, M.A.: Simulating Turing machines with polarizationless P systems with active membranes. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) Membrane Computing: 15th International Conference, LNCS vol. 8961, 229–240 (2014)

[18] Gensler, H.J.: Introduction to Logic, Routledge, London (2002)

[19] Gutierrez-Naranjo, M.A., Perez-Jimenez, M.J., Riscos-Núñez, A., Romero-Campero, F.J.: On the power of dissolution in P systems with active membranes. In: Freund, R., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing: 6th International Workshop, LNCS vol. 3850, 224–240 (2006)

[20] Ionescu, M., Păun, G., Yokomori, T.: Spiking Neural P Systems. Fundamenta Informaticae 71(2): 279–308 (2006)

[21] Johnson, D.S.: A Catalog of Complexity Classes, Algorithms and Complexity (CHAPTER 2), Elsevier, 67-161 (1990)

[22] Jones, N.D.: Space-bounded reducibility among combinatorial problems. Journal of Computer and System Sciences, 11(1):68–85 (1975)

[23] Krishna, S.N., Rama, R.: A variant of P systems with active membranes: Solving NP-complete problems. Romanian Journal of Information Science and Technology, 2, 4, 357–367 (1999)

[24] Leporati, A., Ferretti, C., Mauri, G., Pérez-Jiménez, M.J., Zandron, C.: Complexity aspects of polarizationless membrane systems. Natural Computing **8**(4):703-717 (2009)

[25] Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Simulating elementary active membranes, with an application to the P conjecture. In: M. Gheorghe, G. Rozenberg, A. Salomaa, P. Sosík, C. Zandron (Eds.) Membrane Computing – 15th International Conference, CMC15, LNCS vol. 8961, 284–299 (2014)

[26] Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Monodirectional P systems. Natural Computing, 15, 551–564 (2016)

[27] Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Solving a special case of the P conjecture using dependency graphs with dissolution. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing: 18th International Conference, LNCS vol. 10725, 196-213 (2017)

[28] Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Characterizing PSPACE with shallow non-confluent P systems. Journal of Membrane Computing 1: 75-84 (2019)

[29] Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: A Turing machine simulation by P systems without charges. Journal of Membrane Computing 2, 71–79 (2020)

[30] Mix Barrington, D.A., Immerman, N., Straubing, H.: On uniformity within $\mathbf{NC}^1$. Journal of Computer and System Sciences 41(3), 274–306 (1990)

[31] Murphy, N., Woods, D.: Active membrane systems without charges and using only symmetric elementary division characterise P. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing: 8th International Workshop, LNCS vol. 4860, 367–384 (2007)

[32] Murphy, N., Woods, D.: A Characterisation of NL Using Membrane Systems without Charges and Dissolution. In: Calude, C.S., da Costa, J.F.G., Freund, R., Oswald, M., Rozenberg, G. (eds.) Unconventional Computing: 7th International Conference, LNCS vol. 5204, 164–176 (2008)

[33] Murphy, N., Woods, D.: On acceptance conditions for membrane systems: characterisations of **L** and **NL**. In Neary, T., Woods, D., Seda, T., Murphy, N., eds.: Proceedings International Workshop on The Complexity of Simple Programs, Cork, Ireland, Volume 1 of Electronic Proceedings in Theoretical Computer Science., Open Publishing Association, 172–184 (2009)

[34] Murphy, N., Woods, D.: The Computational Power of Membrane Systems Under Tight Uniformity Conditions. Natural Computing: an international journal **10**(1), 613–632 (2011)

[35] Murphy, N., Woods, D.: Uniformity is weaker than semi-uniformity for some membrane systems. Fundamanta Informaticae **134**(1-2), 129–152 (2014)

[36] Orellana-Martín, D., Riscos-Núnez, A.: Seeking computational efficiency boundaries: the Păun's conjecture. Journal of Membrane Computing 2: 323-331 (2020)

[37] Orellana-Martín, D., Valencia-Cabrera, L., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Minimal cooperation as a way to achieve the efficiency in cell-like membrane systems. Journal of Membrane Computing **1**, 85–92 (2019)

[38] Papadimitriou, C.H.: Computational Complexity. Addison-Wesley Publishing Company, Inc. (1994)

[39] Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences **61**(1), 108–143 (2000)

[40] Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics **6**(1), 75–90 (2001)

[41] Păun, A., Păun, Gh., Rozenberg, G.: Computing by communication in networks of membranes. International Journal of Foundations of Computer Science **13**(6), 779–798 (2002)

[42] Păun, A., Păun, Gh., Rozenberg, G.: The power of communication: P systems with symport/antiport. New Generation Computing **20**(6), 295–305 (2002)

[43] Păun, Gh.: Membrane Computing. An Introduction. Springer (2002)

[44] Păun, Gh.: Further twenty six open problems in membrane computing. In: Third Brainstorming Week on Membrane Computing. Fénix Editora, Sevilla, 249–262 (2005)

[45] Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford, England (2010)

[46] Pérez-Jiménez, M.J., Sancho-Caparrini, F.: A Formalization of Transition P Systems. Fundamenta Informaticae, **49**(1), 261–272 (2002)

[47] Pérez-Jiménez, M.J., Romero-Jiménez, Á., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing **2**(3), 265–285 (2003)

[48] Pérez-Jiménez, M.J., Romero-Jiménez, Á., Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division. Journal of Automata, Languages and Combinatorics **11**(4), 423–434 (2006)

[49] Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Sublinear-Space P Systems with Active Membranes. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds) Membrane Computing, 13th International Conference, LNCS vol. 7762, 342-357 (2013)

[50] Sosík, P.: The computational power of cell division in P systems. Natural Computing **2**(3), 287–298 (2003)

[51] Sosík, P.: P systems attacking hard problems beyond NP: a survey. Journal of Membrane Computing 1, 198–208 (2019)

[52] Sosík, P., Rodríguez-Patón, A.: Membrane computing and complexity theory: A characterization of PSPACE. Journal of Computer and System Sciences **73**(1), 137–152 (2007)

[53] Trudeau, R.J.: Introduction to Graph Theory. Dover Publications. (2013)

[54] Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del-Amor, M.A., Pérez-Jiménez, M.J., Riscos-Núñez A.: Polarizationless P systems with active membranes: Computational complexity aspects. J. Autom. Lang. Comb. **21**(1–2), 107–23 (2016)

[55] Valencia-Cabrera, L., Orellana-Martín, D., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Counting Membrane Systems. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Zandron, C. (eds) Membrane Computing. CMC 2017. Lecture Notes in Computer Science, vol 10725. Springer, Cham. (2018)

[56] Woods, D., Murphy, N., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Membrane dissolution and division in P. In: Calude, C.S., da Costa, J.F.G., Dershowitz, N., Freire, E., Rozenberg, G. (eds.) Unconventional Computation: 8th International Conference, LNCS vol. 5715, 262–276 (2009)

[57] Zandron, C.: Bounding the space in P systems with active membranes. Journal of Membrane Computing 2: 137-145 (2020)

[58] Zandron, C., Ferretti, C., Mauri, G.: Solving NP-complete problems using P systems with active membranes. In: Unconventional Models of Computation, UMC'2K: Proceedings of the Second International Conference on Unconventional Models of Computation. Springer London, London, 289–301 (2001)

Summary

This PhD thesis presents some results on the computational power of certain variants of polarizationless P systems with active membranes. P systems with active membranes are formal computational models designed to simulate processes in living cells. These P systems work on a membrane structure that consists of hierarchically embedded membranes. We consider this structure as a tree in which the nodes are membranes and the directed edges define the parent-child relation between the membranes. Moreover, each membrane has a label and a polarization: positive ($+$), negative ($-$) or neutral (0). The leaves of the tree are called elementary membranes. The membranes delimit membrane regions that may contain some objects representing chemicals from biology. A configuration of the P system is the membrane structure with multisets of objects assigned to its membranes. The objects may evolve according to a given set of rules. Basically, P systems with active membranes operate with evolution rules, in- and out-communication rules, membrane dissolution rules, and division rules for elementary membranes. In many cases, division for non-elementary membranes is also considered. In each time unit, some of the rules are applied. In this way, the P system can perform transitions from one configuration to another. A sequence of these transitions is called computation, and such a transition is called computation step. During a computation step the rules are applied nondeterministically according to the concepts of maximal parallelism. A computation is halting if it reaches a configuration in which no rules can be applied. A P system is a recognizer P system if each of its computations halts, it has two designated objects yes and no, it has designated input and output membranes, and each of its computations must produce exactly one yes or no (but not both) in the output membrane, exactly in the last step of the computation. A P system is confluent if all halting computations on the same input must produce the same output. In the thesis, we deal only with the polarizationless variant of confluent P systems with active membranes where each membrane has the same polarization and these polarizations never change. Polarizationless P systems with active membranes are able to solve every problem in $\mathbf{PSPACE}$ efficiently if division for non-elementary membrane division rules are allowed. We often refer to Păun's conjecture which is one of the key elements in the thesis and roughly sounds as follows. Polarizationless P systems with active membranes cannot solve computationally hard problems

in polynomial time without non-elementary membrane division rules.

In Chapter 2, we discuss the basic necessary topics such as multisets, graphs, propositional logic, and some relevant complexity classes. Then, we explicitly define polarizationless P systems with active membranes and demonstrate a computation step in detail focusing on how the rules are chosen to be applied according to maximal parallelism. Moreover, we define various uniformity conditions for P systems, as in membrane computing, problems are usually solved by uniform families of P systems. A family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ of P systems is $\mathbf{P}$-uniform (resp., $\mathbf{L}$-uniform) if there is a deterministic Turing machine working in polynomial time (resp., using logarithmic space) that computes a reasonable representation of $\Pi(n)$ whenever it is started with 1^n on its input tape and computes the encoding of the input. Sometimes we consider a more restrictive uniformity condition called $\mathbf{DLOGTIME}$-uniformity. Since deterministic log-time Turing machines are not able to compute this representation and the encoding of the input, we define $\mathbf{DLOGTIME}$-uniformity in a different way, as follows. $\mathbf{\Pi}$ is $\mathbf{DLOGTIME}$-uniform if there exists a deterministic log-time Turing machine that recognizes a certain descriptive language of $\mathbf{\Pi}$ and the input.

We investigate the power of dissolution rules in Chapter 3. It is known that polarizationless P systems with active membranes working in polynomial time are not able to solve any problem beyond $\mathbf{P}$ if the use of dissolution rules is not allowed. Moreover, it was also shown that using tighter uniformity conditions results in a weakening of the computational power of these P systems. This gave us motivation to study what polynomial-time polarizationless P systems with active membranes are capable of when they can apply only dissolution rules. We call these P systems dissolution P systems. In the chapter, we deal with a variant of the $\mathbf{NL}$-complete REACHABILITY problem. In this variant, topologically sorted directed graphs are considered and the task is to decide whether there is a path from a node u to node a v. This form of REACHABILITY is still $\mathbf{NL}$-complete. Our result in this chapter, presented in Theorem 1, is the following: the REACHABILITY problem can be solved efficiently by an $\mathbf{L}$-uniform family of dissolution P systems. This means that these systems can solve all problems in $\mathbf{NL}$, that is, $\mathbf{NL}$ is a lower bound of the computational power of dissolution P systems.

We continue the investigation of the computational power of dissolution P systems in Chapter 4. In the first half of the chapter, we show that dissolution P systems are able to solve $\mathbf{P}$-complete problems efficiently under $\mathbf{L}$-uniform conditions. We consider a $\mathbf{P}$-complete variant of the HORNSAT problem called HORN3SATNORM. The instances of this problem can contain only clauses that are either positive unit clauses or they are either of the forms $\neg x \vee \neg y$ or $\neg x \vee \neg y \vee z$. The solution of HORN3SATNORM is inpired by that of REACHABILITY discussed in Chapter 3. In the second part of the chapter, we present a method for modifying the constructed P systems properly to define $\mathbf{DLOGTIME}$-uniform conditions for them. Thus, in summary, the main re-

sult of the chapter is Theorem 2 which says that HORN3SATNORM can be solved in polynomial time by a **DLOGTIME**-uniform family of dissolution P systems. Defining **DLOGTIME**-uniformity for these P systems means that HORN3SATNORM is undoubtedly solved by the P systems and not by the Turing machine. With this result, we give a **P** lower bound to the computational power of polynomial-time polarization P systems with active membranes using only dissolution rules. Since **P** is also an upper bound of the computational power of these P systems, in this chapter, we define a characterization of **P** by dissolution P systems working in polynomial time under rather tight uniformity conditions. This means that considering also evolution rules, communication rules and division rules for elementary membranes does not raise the computational power of the resulting P system on the condition that Păun's conjecture is true.

In Chapter 5, we deal with polarizationless P systems with active membranes using so-called weak non-elementary division rules of the form $[a]_h \rightarrow [b]_h[c]_h$. These rules are generalizations of the elementary membrane division rules. It is known that polynomial-time polarizationless P systems with active membranes are able to solve the **PSPACE**-complete QSAT problem even without in-communication rules, and using out-communication rules only in the last step of the computation to send the answer of the P system out to the environment. Moreover, the employed strong non-elementary membrane division rules are also restricted as they have the form $[[\,]_{h_1}[\,]_{h_1}]_{h_0} \rightarrow [[\,]_{h_1}]_{h_0}[[\,]_{h_1}]_{h_0}$, that is, the inner membranes on the left-hand side of these rules have the same labels. However, it was shown that the upper bound of the computational power of polynomial-time monodirectional P systems is $\mathbf{P}^{\mathbf{NP}}$, which is presumably strictly contained in **PSPACE**. Monodirectional P systems are P systems with active membranes without in-communication rules and using weak division rules for non-elementary membranes. The question arises: How powerful (or weak) are these weak non-elementary membrane division rules? In this chapter, we try to get closer to answering this questions. In Theorem 3, we show that polarizationless P systems with active membranes using **P**-uniformity conditions are able to solve a **PSPACE**-complete variant of QSAT in polynomial time even if they use only in-communication rules, dissolution rules, and weak division rules for non-elementary membranes. In this QSAT variant, the input quantified formula φ is in prenex form with alternating quantifiers, where the first is an existential and the last is a universal quantifier. Since the upper bound of the computational power of polynomial-time P systems with active membranes is **PSPACE**, with this result we give a characterization of this class.

Chapter 6 differs in some aspects from the previous ones in the sense that we focus mainly on investigating certain acceptance conditions defined for P systems working in recognizing mode, that is, used as deciders. In the previous three chapters, we solved the presented problems with recognizer P systems, which is quite

usual. However, in some cases, the acceptance conditions are the only limiting factors in solving certain problems with recognizer P systems. More precisely, these P systems must produce exactly one object yes or no in the output membrane and exactly in the last step of the computations. What if a P system is capable of producing some copies of yes if and only if the input should be accepted, but it cannot reduce the number of these copies to one? Acknowledger P systems also work in recognizing mode, but they are more general than recognizer P systems with respect to their acceptance condition. More precisely, a computation of an acknowledger P system is accepting if and only if the output membrane of its halting configuration contains some copies of yes. An important condition is that object yes cannot occur on the left-hand side of any rule. We even generalize acknowledger P systems by not considering the latter condition. We call these generalized P systems extended acknowledger P systems. We also investigate counting P systems that are devoted to solving counting problems. Moreover, we introduce elimination P systems that are polarizationless P systems with active membranes extended with rules of the form $[ab \rightarrow \varepsilon]_h$, where a and b are objects and ε denotes the empty word. In this chapter, we consider two problems. MAJORITY SAT is a $\mathbf{PP}$-complete problem that sounds as follows: given a Boolean formula φ, decide whether φ is satisfiable by more than half of the possible truth assignments or not. #SAT is a $\#\mathbf{P}$-complete problem, the task is to count all satisfying truth assignments of the input Boolean formula.

Chapter 6 contains five theorems. First, in Theorem 4, we generalize a known result by formally proving that the upper bound of the computational power of $\mathbf{L}$-uniform families of polynomial-time recognizer elimination P systems using no dissolution rules is $\mathbf{NL}$. Then, in Theorem 5, we show that #SAT can be solved efficiently by an $\mathbf{L}$-uniform family of counting elimination P systems without using communication rules. Next, in Theorem 6, we prove that the recognizer variants of these P systems using the same uniformity conditions are able to solve MAJORITY SAT in polynomial time. In Theorem 7, we show that #SAT can be also solved efficiently by an $\mathbf{L}$-uniform family of counting elimination P systems without using dissolution rules. Finally, in Theorem 8, we present an $\mathbf{L}$-uniform family of extended acknowledger elimination P systems using no dissolution rules that solves MAJORITY-SAT in polynomial time. The first and fifth theorems of this chapter imply that extended acknowledger P systems are more powerful than recognizer P systems in terms of computation power.

Contributions of the thesis

In Section 2.2 of **Chapter 2**, the contributions are related to [5].

II/1. The author declares that his contribution to Section 2.2 is the same as that of the co-authors.

In **Chapter 3**, the contributions are related to [2].

III/1. The author declares that his contribution to the design of Algorithm 1 is the same as that of the co-author.

III/2. The author declares that his contribution to Theorem 1 is the same as that of the co-author.

In **Chapter 4**, the contributions are related to [3].

IV/1. The author declares that his contribution to the design of Algorithm 2 is the same as that of the co-author.

IV/2. The author declares that his contribution to the first part Theorem 2 is the same as that of the co-author.

IV/3. The author declares that his contribution to the second part of Theorem 2 (Section 4.5) is decisive.

In **Chapter 5**, the contributions are related to the [1].

V/1. The author declares that his contribution to the design of Algorithm 3 is the same as that of the co-authors.

V/2. The author declares that his contribution to Theorem 3 is the same as that of the co-authors.

In **Chapter 6**, the contributions are related to the [4].

VI/1. The author declares that his contribution to the design of Algorithm 4 is the same as that of the co-author.

VI/2. The author declares that his contribution to the Theorem 4 is less than that of the co-author.

VI/3. The author declares that his contribution to Theorem 5 is the same as that of the co-authors.

VI/4. The author declares that his contribution to Theorem 6 is the same as that of the co-authors.

VI/5. The author declares that his contribution to Theorem 7 is decisive.

VI/6. The author declares that his contribution to Theorem 8 is the same as that of the co-authors.

Összefoglalás

A Phd dolgozatban az aktív membrános P rendszerek számítási erejéről mutatunk be néhány eredményt. Az aktív membrános P rendszerek olyan formális modellek, melyeket az élő sejtek biológia folyamatainak a szimulálására terveztek. Ezek a P rendszerek egy membránstruktúrán működnek, amely hierarchikusan egymásba ágyazott membránokból áll. A struktúrát egy fának tekintjük, amelynek a csúcsai a membránok, az élei pedig a membránok beágyazását definiálják. A fa leveleit elemi membránoknak nevezzük. Továbbá minden membrán rendelkezik egy címkével és egy polarizációval: pozitív ($+$), negatív ($-$) és semleges (0). A membránok membránrégiókat határolnak, melyek objektumok multihalmazát tartalmazzák. A P rendszer egy konfigurációját egy olyan membránstruktúraként definiáljuk, melynek minden membránjához objektumok egy multihalmazát rendeljük. Az objektumok evolválhatnak a megadott szabályok alapján. Az aktív membrános P rendszerek alapvetően evolúciós, befelé kommunikáló, kifelé kommunikáló, membránfeloldó és elemi membránosztó szabályokat alkalmazhatnak. Számos esetben vizsgáltak ugyanakkor nemelemi membránosztó szabályokat alkalmazó P rendszereket is. A P rendszerek minden időegységben alkalmaznak valahány szabályt, így valósíthatóak meg a konfigurációk közötti átmenetek. Egy ilyen átmenetet számítási lépésnek nevezünk, míg átmeneteknek egy sorozatát pedig számításnak. Egy számítási lépés során a szabályok nemdeterminisztikus módon kerülnek alkalmazásra a maximális párhuzamosság elve szerint. Egy számítást terminálónak nevezünk, ha az eljut egy olyan konfigurációba, melyben egyetlen szabály sem alkalmazható. Egy P rendszer felismerő, ha annak minden számítása terminál, rendelkezik a yes és a no kitüntetett objektumokkal, van egy input- és egy output membránja, valamint minden számításának kizárólag az utolsó lépése során a yes és no objektumok közül pontosan az egyiket jeleníti meg az utolsó konfiguráció output membránjában. Egy P rendszert konfluensnek nevezünk, ha ugyanazon az inputon elindítva minden terminál számításának ugyanaz a kimenete. A tézisben kizárólag polarizációmentes aktív membrános P rendszerekkel foglalkozunk. Az ilyen P rendszerekben minden membránnak ugyanaz a polarizációja, amely sohasem változik meg a számítások során. A polarizációmentes aktív membrános P rendszerek minden problémát képesek polinomiális időben megoldani a **PSPACE** osztályon belül, ha nemelemi membránosztás is engedélyezett. Gyakran említjük a dolgozatban a Păun-sejtést, amely nagyjából

a következőképpen szól: a polarizációmentes aktív membrános P rendszerek nem képesek **NP**-teljes problémák hatékony megoldására nemelemi membránosztási szabályok alkalmazása nélkül.

A 2. fejezetben olyan alapvető fogalmakat definiálunk, mint a multihalmazok, a gráfok és az ítéletkalkulus, valamint a releváns bonyolultsági osztályok. Ezután formálisan is definiáljuk a polarizációmentes aktív membrános P rendszereket, melynek keretében részletesen bemutatunk egy számítási lépést arra fókuszálva, hogy hogyan választhatóak ki az alkalmazandó szabályok a maximális párhuzamosság elve szerint. Mivel problémákat általában konfluens P rendszerek uniform családjaival szokás megoldani, definiáljuk a dolgozatban használt, P rendszerekre vonatkozó uniformizálási feltételeket. P rendszereknek egy $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ családja **P**-uniform (rendre **L**-uniform), amennyiben létezik olyan determinisztikus polinomiális idejű (logaritmikus-táras) Turing gép, amely ki tudja kiszámítani $\Pi(n)$-nek egy megfelelő reprezentációját, ha 1^n szerepel az input szalagján, valamint ki tudja számítani az input elkódolását. Olykor viszont sokkal szigorúbb uniformizálási feltételekre is szükségünk lesz, ilyen például a **DLOGTIME**-uniformizálás. Mivel a determinisztikus logaritmikus-idejű Turing gépek nem képesek a P rendszerek reprezentációjának és az input elkódolásának a kiszámítására, a szokásostól eltérően kell definiálnunk a **DLOGTIME**-uniformizálást. Röviden, P rendszereknek egy $\mathbf{\Pi}$ családja **DLOGTIME**-uniform, ha létezik olyan determinisztikus logaritmikus-idejű Turing gép, amely felismeri $\mathbf{\Pi}$-nek és az inputnak egy bizonyos leíró nyelvét.

A 3. fejezetben a membránfeloldó szabályok erejét vizsgáljuk meg. Ismert, hogy a polarizációmentes aktív membrános P rendszerek nem képesek egyetlen problémát sem megoldani hatékonyan a **P** osztályon kívül, amennyiben membránfeloldó szabályok használata nem engedélyezett. Továbbá az is ismert, hogy gyengébb uniformizálási feltételek használatával gyengül ezeknek a P rendszereknek a számítási ereje. Ez adta a motivációt ahhoz, hogy a kizárólag feloldó szabályokat alkalmazó polarizációmentes aktív membrános P rendszereket tanulmányozzuk. Ezeket a P rendszereket a dolgozatban egyszerűen feloldó (dissolution) P rendszereknek nevezzük. A fejezetben az **NL**-teljes ELÉRHETŐSÉG probléma egy változatával foglalkozunk, melyben az input egy topologikusan rendezett, irányított gráf és a kérdés, hogy létezike út egy u csúcsból egy v csúcsba. Ezen probléma ugyanúgy **NL**-teljes, mint az ELÉRHETŐSÉG probléma. A fejezet fő eredménye, az 1. tétel a következőképpen szól: az ELÉRHETŐSÉG probléma fenti változata hatékonyan megoldható a feloldó P rendszereknek egy **L**-uniform családjával. Ez azt jelenti, hogy minden problémát képesek megoldani **NL**-ben, azaz ezen P rendszerek számítási erejének egy alsó korlátját adjuk így meg.

A 4. fejezetben folytatjuk a feloldó P rendszerek számítási erejének vizsgálatát. A fejezet első felében megmutatjuk, hogy a feloldó P rendszerek **L**-uniform családjai képesek **P**-teljes problémák megoldására is polinomiális időben. Ehhez a HORNSAT

probléma egy **P**-teljes változatát, a HORN3SATNORM problémát oldjuk meg ilyen P rendszerekkel. Ezen problémának a példányai csak olyan klózokat tartalmazhatnak, melyek pozitív egységklózok, esetleg $\neg x \vee \neg y$ vagy $\neg x \vee \neg y \vee z$ alakban vannak. A HORN3SATNORM megoldását a 3. fejezetben taglalt ELÉRHETŐSÉG problémára adott megoldás motiválta. A 4. fejezet második felében bemutatunk egy eljárást, amivel úgy módosítjuk a megépített P rendszereket, hogy a **DLOGTIME**-uniformizálási feltételeknek is megfeleljenek. Összegezve, a fejezet fő eredménye, a 2. tétel a következőképpen szól: a HORN3SATNORM probléma hatékonyan megoldható feloldó P rendszereknek egy **DLOGTIME**-uniform családjával. A **DLOGTIME** Turing gépek limitált kifejezőereje biztosítja, hogy a HORN3SATNORM problémát valóban a P rendszerek oldják meg és nem a definiálásukhoz használt **DLOGTIME** Turing gép. Ezzel az eredménnyel definiálunk egy **P** alsó korlátot a polinomiális idejű feloldó P rendszerek számítási erejére vonatkozóan. Így, mivel ezen P rendszerek számítási erejére **P** egy felső korlát is egyben, sikerül jellemeznünk a **P** osztályt polinomiális idejű, nagyon gyenge uniformizálási feltételekkel bíró feloldó P rendszerek segítségével. Ez azt is jelenti, hogy az evolúciós, a kommunikációs és az elemi membránosztó szabályok hozzáadásával a kapott P rendszer számítási ereje nem nő, amennyiben a Păun-sejtés valóban igaz.

Az 5. fejezetben olyan polarizációmentes aktív membrános P rendszerekkel foglalkozunk, melyek az $[a]_h \rightarrow [b]_h[c]_h$ alakú, gyenge (weak) nemelemi membránosztó szabályokat is használhatják. Ezek a szabályok az elemi membránosztó szabályoknak a kiterjesztései nemelemi membránokra. Ismert, hogy a polinomiális idejű polarizációmentes aktív membrános P rendszerek meg tudják oldani a **PSPACE**-teljes QSAT problémát befelé kommunikáló szabályok nélkül is, $[[\ ]_{h_1}[\ ]_{h_1}]_{h_0} \rightarrow [[\ ]_{h_1}]_{h_0}[[\ ]_{h_1}]_{h_0}$ alakú nemelememi membránosztó szabályok segítségével (vagyis a szabály bal oldalán a belső membránok címkéi mindig megegyeznek). Ráadásul úgy, hogy kifelé kommunikáló szabályokra is csak az utolsó lépésben van szükség az output kiküldésére a környezetbe. Azonban az is ismert, hogy a polinomiális idejű, egyirányú (monodirectional) P rendszerek számítási erejének egy felső korlátja $\mathbf{P}^{\mathbf{NP}}$, ami feltehetően **PSPACE** valódi részhalmaza. Ezek az egyirányú P rendszerek olyan aktív membrános P rendszerek, melyek nem használnak befelé kommunikáló szabályokat, de gyenge nemelemi osztást viszont igen. Felmerül a kérdés: Mennyire erősek (vagy éppen gyengék) ezek a gyenge nemelemi membránosztási szabályok? Ebben a fejezetben ennek a kérdésnek a megválaszolásához próbálunk közelebb kerülni. A 3. tételben bebizonyítjuk, hogy a polarizációmentes aktív membrános P rendszerek **P**-uniform családjai képesek hatékonyan megoldani a QSAT problémát kizárólag befelé kommunikáló, feloldó és gyenge nemelemi membránosztó szabályokat használva. Mivel ezen rendszerek számítási erejének egy felső korlátja **PSPACE**, ezért az eredményünk a **PSPACE** egy újabb jellemzése.

A 6. fejezet tartalmilag kissé eltér a korábbiaktól, ezúttal elsősorban a felis-

merő módban működő P rendszerekre megfogalmazott bizonyos elfogadási feltételek kerülnek a középpontba. Az előző három fejezetben a kérdéses problémákat felismerő P rendszerekkel oldottuk meg, ami teljesen szokványos a membránszámításban. Azonban vannak olyan esetek, amikor bizonyos problémák eldöntésében a korlátozó tényező éppen az elfogadási feltételekben keresendő. Konkrétan, a felismerő P rendszerekben a számításoknak pontosan az utolsó lépésében a yes vagy a no objektumokból pontosan az egyiknek meg kell jelennie az output membránban. Mi van akkor, ha a P rendszer képes elérni, hogy legalább egy yes objektum megjelenjen az output membránban, de arra már nem képes, hogy ezeknek az objektumoknak a számát egyre redukálja? A nyugtázó (acknowledger) P rendszerek ugyancsak felismerő módban működnek, de a felismerő P rendszereknél általánosabbak, ami az elfogadási feltételeket illeti. Pontosabban, egy nyugtázó P rendszernek egy számítása akkor és csak akkor elfogadó, ha az utolsó konfigurációjában az output membrán tartalmaz legalább egy yes-t. A nyugtázó rendszerekben a yes nem szerepelhet egyik szabály bal oldalán sem. Mi ezt a feltételt nem kötjük ki, ezzel definiálva a nyugtázó P rendszereknek egy általánosítását, melyeket kiterjesztett nyugtázó P rendszereknek nevezünk. Ezen felül a fejezetben foglalkoztunk még számláló P rendszerekkel, amelyeket számlálási problémák megoldására használnak. Bevezetjük továbbá az eliminálló (elimination) P rendszereket, melyek olyan polarizációmentes aktív membrános P rendszerek, melyek $[ab \to \varepsilon]_h$ alakú eliminálló szabályokat is alkalmazhatnak. Ebben a fejezetben két problémára adunk megoldást eliminálló P rendszerekkel. A MAJORITY-SAT probléma egy **PP**-teljes probléma és a következőképpen szól: igaz-e, hogy egy adott ítéletkalkulusbeli φ formulához tartozó értékadásoknak több, mint a fele kielégítő? A másik probléma, a #SAT egy $\#\mathbf{P}$-teljes számlálási probléma, melynél a feladat az input ítéletkalkulusbeli formula összes kielégítő értékadásának a megszámolása.

A 6. fejezet öt tételt tartalmaz, a taglalt megoldásokban végig **L**-uniformizálási feltételeket definiálunk. A 4. tételben általánosítunk egy már ismert eredményt: formális bizonyítást adunk arra, hogy a feloldó szabályokat nem alkalmazó, polinomiálisidejű felismerő eliminálló P rendszereknek a számítási erejére **NL** egy felső korlát. Ezután az 5. tételben megmutatjuk, hogy #SAT hatékonyan megoldható kommunikációs szabályokat nem használó számláló eliminálló P rendszerekkel. Majd a 6. tételben bebizonyítjuk, hogy ezen P rendszerek felismerő változatai hatékonyan meg tudják oldani a MAJORITY-SAT problémát. Ezt követően a 7. tételben megmutatjuk, hogy #SAT ugyancsak megoldható hatékonyan számláló eliminálló P rendszerekkel, ezúttal feloldó szabályok használata nélkül. Végül a 8. tételben azt mutatjuk meg, hogy a MAJORITY-SAT probléma hatékonyan megoldható olyan kiterjesztett nyugtázó eliminálló P rendszerkkel, melyek nem alkalmaznak feloldó szabályokat. A 4. és a 8. tételekből következik, hogy a feloldó szabályok nélküli kiterjesztett nyugtázó P rendszerek számítási ereje valóban nagyobb, mint a felismerő P rendszereké.

Acknowledgments

First of all, I would like to thank my supervisor, Dr. Zsolt Gazdag, for his dedicated work over the years. He is one of the most conscientious and helpful people I know. There were some factors (Covid-19, family duties, etc.) that sometimes made our work a little difficult, but nevertheless we managed to bring this story to a successful conclusion. And hopefully, we will continue to work together in the future.

I am very grateful to Professor Zoltán Fülöp, my supervisor during my BSc and MSc studies, for introducing me to computer science. Without him, I would never have made it anywhere near the PhD level.

I have to mention Dr. Szabolcs Iván, my other coauthor, who I worked with mainly in the early years. It was good to brainstorm together, although unfortunately it did not happen very often.

Last but not least, I wish to thank my parents who stood by me during the most difficult times when my life was in ruins. Their constant support helped me a lot in returning to the university and achieving this accomplishment.